Automatic Guidance of Agricultural Wide-Span Implement Carrier (WSIC)

by

Chengming Luo

Thesis submitted in partial fulfillment of the requirements for the degree of
Doctorate in Philosophy
in Mechanical Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
Faculty of Engineering
University of Ottawa

Abstract

The Wide-Span Implement Carrier (WSIC) is a versatile agricultural platform for the support and operation of different agricultural field equipment. It has been used in cranberry production since the 1990s. The main components of the WSIC include a long truss and two low-profile tractors that provide support to the truss at each end. WSIC operations require the motions of the two tractors to be well synchronized. Currently, this process is controlled by human operators on the tractors and the guidance accuracy depends on their experience and on-site judgement.

The objective of this research is to develop an automatic guidance system to replace the manual guidance and control process of the WSIC, so that the demand of operators can be reduced and the operation accuracy can be improved. The original contributions of this work are the development and experimental validation of the two groups of control algorithms designed for the WISC, one for its mobile operating mode and one for its stationary operating mode. The mobile mode was modelled as a synchronous trajectory tracking problem and a master-slave cooperative method was designed. The stationary mode was modelled as a parallel point-to-point tracking problem and a second master-slave cooperative method was designed. For the mobile mode, the master vehicle performs a path following task by controlling its steering angle, and the slave vehicle performs a path following task and a synchronous tracking task by controlling both the steering angle and the velocity. During the operation, the motion states of the master including its position, orientation, and velocity are sent to the slave in real time. For the stationary mode, both vehicles perform a point tracking task and repeat the task in a periodic control sequence, which is executed on the master. The slave follows the commands sent from the master to perform point tracking processes. The designed control algorithms were first verified through several simulations.

To test the control models and algorithms in practical applications, a scaled-down experimental WSIC platform using two heavy-duty mobile robots and an original automatic guidance system adapted to WSIC operations were developed. The hardware and software of the guidance system were designed and developed. The hardware included a dual-rover Real-Time Kinematic Global Positioning System, inertial measurement units, XBee wireless communication modules, and a group of control processors. The control structure of the software was developed at two levels: upper level for guidance algorithms calculation and lower level for velocity and steering angle control.

Validation experiments were conducted using the robotic platform under restricted conditions including flat and firm test grounds and reasonably slow operation velocities. For each operating mode, two series of experiments were performed. For the mobile mode, the

first series tested the path following control for a single robot, and the second series tested the cooperative synchronous tracking control. For the stationary mode, the first series verified the designed velocity and steering angle control laws for the point tracking task, and the second series tested the sequential point-to-point tracking control. Experimental results showed that the developed guidance system performed with satisfactory accuracy. For the mobile mode, the master and slave robots could automatically follow the reference paths with lateral errors less than 0.1 m and orientation errors less than 5°, and the motions of the two robots were well synchronized with offset errors less than 0.1 m. For the stationary mode, the robots performed the forward point tracking tasks with lateral errors less than 0.16 m and orientation errors less than 11°, and the root mean square and maximum of the offset errors were 0.04 m and 0.1 m respectively. The time differences when the robots stopped at their target points were less than 2 s. This motion synchronization could permit high efficiency for autonomous WSIC operations.

Acknowledgements

This work would not have been possible without the guidance, help, and support of many people.

First, I would like to express my heartfelt gratitude to my supervisor, Dr. Claude Laguë, for his guidance, support, enthusiasm, encouragement, and patience throughout the course of my Ph.D. study. His professionalism, generosity, positivity, and sense of humour have influenced me deeply and will continue to benefit me in my future work and life.

I want to express my sincere appreciation to my colleague and friend, Dr. Ahmad Mohsenimanesh, for all the timely help, support, and encouragement that he selflessly provided to me.

I would like to cordially thank Dr. Neil McLaughlin (Research scientist at Agriculture and Agri-Food Canada), Dr. Dan Necsulescu (Professor in the Department of Mechanical Engineering, University of Ottawa), and Dr. Jie Liu (Associate Professor in the Department of Mechanical and Aerospace Engineering, Carleton University) for serving as my committee members and providing important and insightful suggestions to my research project. Many thanks to Dr. Pierre Payeur (Professor in the School of Electrical Engineering and Computer Science, University of Ottawa) and Dr. Mohammad Biglarbegian (Associate Professor in the School of Engineering, University of Guelph) for evaluating this thesis and providing extensive comments and suggestions for improving it.

Special thanks go to Dr. Colin Rennie, Chair and Professor of the Department of Civil Engineering, University of Ottawa, for generously allowing me to use the RTK-GPS equipment of his laboratory quite a few times and for quite long periods.

I also thank the technicians in the mechanical engineering workshop, John Perrins and Stanley Weedmark, for their work in preparing some of my important experimental equipment. I would particularly like to mention Leo Denner for providing prompt and invaluable technical support throughout this project. His expertise in electronics helped me a great deal. I also thank Susan Rennie for her kind administrative assistance.

I would like to thank the China Scholarship Council and the University of Ottawa for the four-year joint scholarship that I received for my Ph.D. program and the Department of Mechanical Engineering for the teaching and research assistantships that I received during my Ph.D. studies. I would also like to acknowledge Natural Sciences and Engineering Research Council of Canada for funding the research project.

During the past few years in Ottawa, I was surrounded and supported by many friends. I am particularly grateful to Deliang Guo and Lu Han for always being by my side through

the ups and downs of my life and study. I would also like to thank Licheng Peng, Yu Wang, Shuangyin Ren, Shanshan Ren, Rong Ran, Leqiang Sun, Liang Cui, Guang Xu, Zhiliang Yang, Qiang Xiao, Jing Wang, and Huan Huang for their friendship and support.

Last but not least, I wish to offer my utmost gratitude to my parents for their unceasing love, endless patience, and great sacrifice. I would also like to thank my other family members including my uncle, my brother, my sister, my sister-in-law, my brother-in-law, my niece, and my little nephew for being supportive and caring throughout this journey.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

2D	two-dimensional
3D	three-dimensional
ADC	analog to digital converter
CP-DGPS	Carrier-Phase Differential Global Positioning System
CTF	controlled-traffic farming
DC	direct current
DGPS	Differential Global Positioning System
DR	dead reckoning
ENU	east-north-up
FOG	fiber optic gyroscope
FPID	forward proportional-integral-derivative
GPS	Global Positioning System
GUI	graphical user interface
IMU	inertial measurement unit
ISM	industrial, scientific, and medical
LMS	laser measurement system
LRF	laser range finder
MEMS	micro-electromechanical system

NMEA	National Marine Electronics Association
PID	proportional-integral-derivative
RMS	root mean square
RTK-GPS	Real-Time Kinematic Global Positioning System
SD	standard deviation
SPI	serial peripheral interface
TTL	transistor-transistor logic
WSIC	Wide-Span Implement Carrier

Mathematical Symbols (with units at the end)

α'	modified heading error in polar coordinates	$^\circ$
α	heading error in polar coordinates	$^\circ$
β'	modified orientation error in polar coordinates	$^\circ$
β	orientation error in polar coordinates	$^\circ$
δ	steering angle of the virtual front wheel	$^\circ$
δ_l	steering angle of the left wheel	$^\circ$
ψ	angle of the distance line with respect to OX	$^\circ$
ρ'	modified distance error	m
ρ	distance error in polar coordinates	m
ρ_0	initial distance error	m
θ_{pm}	orientation error of the master vehicle	$^\circ$
θ_{ps}	orientation error of the slave vehicle	$^\circ$
θ	vehicle orientation angle	$^\circ$
θ_d	vehicle orientation angle of the desired position	$^\circ$
θ_e	orientation error of the vehicle with respect to the reference vehicle	$^\circ$

θ_p	orientation error of the vehicle with respect to the reference path	$^\circ$
θ_r	orientation angle of the reference vehicle	$^\circ$
θ_t	tangent angle of path C at point M with respect to OX	$^\circ$
b	vehicle tread width	m
C	reference path for the vehicle to follow	
$c(s)$	curvature of path C	1/m
d	lateral error	m
e	offset error	m
k_e	positive control gain	
k_u	positive control gain	
k_v	positive velocity coefficient	m/s
$k_{1,2}$	positive control gains	
$k_{p,i,d}$	positive control gains	
l	vehicle wheelbase	m
M	reference point on path C	
n	positive even order number	
O	origin of the local Cartesian coordinate system	
o	origin of the vehicle body frame	
s	curvilinear coordinate of the reference point along C	m
u_1	virtual control input	
u_{1d}	desired control input	
v	vehicle linear velocity	m/s
v_m	linear velocity of the master vehicle	m/s
v_r	linear velocity of the reference vehicle	m/s

v_s	linear velocity of the slave vehicle	m/s
$V_{1,2}$	Lyapunov functions	
X	local Cartesian coordinate axis	
x	vehicle coordinate along OX	m
x_d	vehicle coordinate of the desired position along OX	m
x_r	coordinate of the reference vehicle along OX	m
Y	local Cartesian coordinate axis	
y	vehicle coordinate along OY	m
y_d	vehicle coordinate of the desired position along OY	m
y_r	coordinate of the reference vehicle along OY	m

Chapter 1

Introduction

1.1 Context and motivation

Modern agriculture faces various challenges such as growing food needs, decreasing agricultural population, and increasing energy and labour costs. Modern agricultural activities need to consider comprehensively economic, energy, environmental, and health and safety aspects. Automatic guidance of agricultural machinery plays an increasingly important role in modern agriculture. Technologies including precise position-and-motion sensing and modern control are introduced into traditional agricultural vehicles to develop automatic guidance systems. As production costs rise and fuel and labour become more expensive, agricultural machines have become larger and more powerful in order to achieve maximum efficiency. The assistance of automatic guidance technologies improves the accuracy and productivity of agricultural operations. Labour force inputs are reduced and the timeliness of critical operations is enhanced. Meanwhile, various automation technologies and equipment allow agricultural producers to operate and manage machines more easily and comfortably, which improves their health and safety.

Over the course of the past two decades, tractors, as the most common agricultural power units, have been studied substantially in the area of automatic guidance and navigation. Self-propelled combine harvesters, transplanters, and orchard vehicles have been taken as common research platforms as well. The guidance problems of these conventional agricultural machines have been successfully solved by advanced technologies such as Real-Time Kinematic Global Positioning System (RTK-GPS), machine vision, and laser. Commercial products are available on the market for easy integration and implementation on these platforms, e.g., the RTK-GPS based guidance and steering system from Trimble Navigation, USA, and the CAMPilot vision system from CLAAS, Germany.

From the 1980s on, the concept of controlled-traffic farming (CTF) has been applied increasingly in agricultural systems to improve the economics and sustainability of agriculture. CTF works on the principle that crop zones and traffic lanes of agricultural machines are distinctly and permanently separated, thus reducing traffic-induced soil compaction (Monroe & Burt, 1989). Benefits of CTF include improved soil health and crop yields, improved timeliness of field operations, and reduced production inputs and costs. Three different platforms have been researched to implement the controlled-traffic concept. Conventional farm tractors were used in early studies, and they provided a maximum width of the traffic-free crop zone of about 3 m (Williford, 1980). Later on, specially-built, wide wheel-tread machines referred to as gantries were developed (Chamen et al., 1992; Tillett et al., 1988). Gantries usually spanned 5 to 10 metres and hence provided a wider traffic-free crop zone than conventional tractors. The third type of platform was named Wide-Span Implement Carrier (WSIC), which typically consisted of a long truss (30 m or more) supported at both ends by drive wheels that ran on permanent lanes, thus providing a much higher non-trafficked to trafficked area ratio.

The WSIC is a versatile agricultural platform able to support and operate different agricultural equipment. In 1997, a prototype WSIC was designed, built, and field-tested to complete the pruning, fertilizing, weeding, and harvesting operations in cranberry production (Laguë et al., 1997). The main components of this WSIC are a 61-m long truss and two low-profile tractors that provide support to the truss at each end. A mobile carriage is mounted under the truss to support and operate different implements. One tractor is connected to the truss by means of three pivot joints which allow the tractor to rotate 360° about the truss. The other tractor is connected to the truss using three pivot and one sliding joints. The additional sliding joint allows the tractor to slide under the truss within a 12.2-m range, which is necessary for the WSIC to accommodate different cranberry field widths. Cranberry fields are normally arranged into regular-shaped rectangles having 46~60 m in width and several hundred metres in length and are very well adapted to WSIC operations. A typical cranberry farm during harvesting season is shown in Figure 1.1. The WSIC has two operating modes: stationary and mobile. Under the stationary mode, the field operations are completed along the span of the main truss while both tractors are at rest. The machine travels forward by a distance corresponding to the working width of the implement after the mobile carriage has completed one pass across the field. The mobile mode involves completion of the field operations in a direction perpendicular to the truss. Under this mode, the implement is positioned at a given location along the span of the main truss and the entire machine is displaced along the length of the field. Figure 1.2 shows (a) the harvesting operation under the stationary mode and (b) the fertilizing operation under the mobile mode.

Figure 1.1: Cranberry farm in Manseau, Québec.

(a) Harvesting in stationary mode

(b) Fertilizing in mobile mode

Figure 1.2: WSIC system for cranberry production.

As a large-size platform, operating the WSIC is labour demanding. Three operators are normally needed, one on each tractor and one on the mobile carriage. Navigating the WSIC is a demanding task as well. Under the stationary mode, both tractors need to move forward by a certain distance and stop at a particular point on the path after the implement finishes one pass across the field. This process is controlled by the two operators on the tractors and the accuracy depends on their experience and on-site judgement. Under the

mobile mode, each operator drives one tractor and keeps it on the desired path. Meanwhile, the two operators need frequent communication in order to guarantee that the two ends of the machine are travelling at the same pace. Previous studies on agricultural automatic guidance using high-accuracy sensors and advanced control and automation technologies verified that centimetre-level guidance accuracy could be achieved (Bayar et al., 2015; Fang et al., 2006; Rovira Más et al., 2010; Q. Zhang & Qiu, 2004). However, it can be difficult to directly apply those well-developed technologies or commercial guidance products on the WSIC, since it features a special construction and its two operating modes are different from operating behaviours of conventional farm vehicles. A crucial requirement for WSIC guidance is that the two tractors should be kept in synchronized motions, and this is particularly important for the mobile operating mode. The periodic stop-go-stop pattern of the tractors under the stationary mode also requires a guidance model that has not been discussed in existing research.

1.2 Research objectives

Notwithstanding all the research activities already performed on automatic guidance of farm vehicles, the guidance problems for the WSIC are not yet solved. Previous investigations focused mainly on independent farm tractors and robots, and the guidance tasks were for those independent platforms to continuously follow straight or curved paths generated during the operation or make headland turns. The WSIC platform requires two parallel paths that are preplanned according to the sizes and shapes of the fields. Some research activities dealt with the problem of tractor-tractor cooperation and master-slave systems were developed (Lee et al., 2007; Noguchi et al., 2004; X. Zhang et al., 2010). In those developments, an autonomous slave tractor was controlled to follow the trajectories of a master tractor, either parallelly or with some determined offset. This kind of cooperation presents certain similarities to the WSIC system. However, in the existing master-slave systems, the trajectories for both tractors were randomly generated and the synchronization requirement was not as strict as for the WSIC. More importantly, the two operating modes of the WSIC require two different control methods that are not directly available from those master-slave systems.

The overall objective of this research is to develop an automatic guidance system to replace the manual guidance and control process of the WSIC, so that the demand of operators can be reduced and the operation accuracy can be improved. Developing such a system includes the following tasks: developing control models and algorithms for the two operating modes, determining the appropriate guidance technologies, and developing sys-

tem hardware and software. The resulting system needs to address the following guidance and control problems for the WSIC:

- To control the two tractors in the mobile mode, making them follow the reference paths while at the same time keeping their motions synchronized;
- To control the two tractors in the stationary mode, making them perform the periodic stop-go-stop processes in a synchronized manner;
- To determine the appropriate guidance technologies and to develop the hardware and software for the physical guidance system; and
- To test the control methods for both operating modes in practical experiments and to evaluate the performance of the guidance system.

1.3 Outline of the thesis

The thesis is structured as follows:

Chapter 1 is the general introduction.

Chapter 2 is the literature review on automatic guidance of agricultural machinery and motion control of mobile robots and vehicles.

Chapter 3 presents the development of control models and algorithms for the two operating modes.

Chapter 4 presents the simulation results for the designed control algorithms. The theoretical effectiveness of the control designs is verified.

Chapter 5 describes the experimental setup for practical validations. An experimental WSIC platform and the corresponding hardware and software of the automatic guidance system are developed. In addition, the experimental site and plans are introduced.

Chapter 6 presents the results of the validation experiments. The effectiveness of the control algorithms are verified and the performance of the automatic guidance and control system is evaluated.

Chapter 7 summarizes the contributions and limitations of this research and suggests directions for future studies.

Chapter 2

Background and Literature Review

The field operation of agricultural vehicles and machines for applications such as tillage, seeding and planting, fertilizing, and harvesting represents an important time-and-energy consuming task. In order to increase the efficiency of multiple agricultural operations and to improve the health and safety of operators, researchers have spent great efforts on developing automatic guidance systems for agricultural vehicles. Those efforts varied from using various promising sensors for precise positioning and navigation to developing suitable vehicle models and control algorithms for automatic guidance purposes.

Motion control is an important task in robotics. Considerable research has been carried out in designing novel control laws for control tasks under different circumstances, such as navigating unmanned vehicles off-road or underwater and controlling robot arms and end effectors to complete fruit harvesting operations. For mobile robots and automatic-guided ground vehicles, three common motion tasks, point stabilization, path following, and trajectory tracking, have been studied through various modern control methods. An automatic-guided agricultural vehicle is a mobile robot that is designed to accomplish agricultural operations under automatic steering and driving. Compared with mobile robots researched in industrial and military sectors, the motion tasks for agricultural robots can be simpler because the latter are mainly required to travel along straight lines on regular-shaped farmlands. However, precise guidance of an agricultural robot can be challenging as many factors influence the overall performance such as wheel skidding and slipping, undulation of farmlands, and inaccurate vehicle models.

In this chapter, some background knowledge and a literature review on automatic guidance of agricultural machinery and motion control of mobile robots and vehicles are presented. Some related works are introduced for each topic.

2.1 Automatic guidance of agricultural machinery

The earliest "driverless tractor" prototypes using leader cable guidance systems date back to the 1950s (Morgan, 1958). At the beginning, techniques applied to automatic guidance of agricultural machines included cable leading, dead reckoning (DR), and laser leading. In the 1980s, the potential for combining computers with image sensors provided opportunities for machine vision based guidance systems. From the 1990s on, the development and deployment of the Global Positioning System (GPS) has driven and accelerated the automatic guidance research in various domains, especially agriculture. The most commonly used technologies for automatic guidance nowadays include GPS, machine vision, laser, and radar. It is a popular trend to combine different kinds of guidance sensors together by applying sensor fusion methods to achieve guidance systems adaptive to different working environments and application requirements.

2.1.1 Framework and methodologies

Figure 2.1 shows a framework of an automatic guidance system, in which the key elements are guidance sensors, computing algorithms, tracking methods, vehicle motion models, and controllers (M. Li et al., 2009; Reid et al., 2000). The most common automatic guidance systems are either GPS based or vision based, featuring absolute positioning or relative positioning. Laser ranger finder (LRF) is another type of relative positioning sensor that can be used to develop guidance systems. DR sensors and inertial sensors, such as speed encoders, odometers, gyroscopes, and accelerometers, are commonly used as auxiliary appliances to a GPS- or vision-based guidance system.

The first phase of an automatic guidance system is sensor measurements. The position, heading, and motion states of the vehicle are determined based on guidance sensor measurements. This information can be conveniently calculated from the measurements of GPS receivers, speed sensors, and inertial measurement units (IMUs). A computing algorithm is mainly used for image processing and guidance signal extraction in a vision system, in which guidance information cannot be directly achieved. Many computing algorithms have been applied in vision-based guidance systems including Hough transform (Ji & Qi, 2010), K-means clustering algorithm (Han et al., 2004), and fuzzy logic (Benson et al., 2003). A guidance system normally contains a group of sensors including GPS receivers or vision cameras, speed encoders, and gyroscopes. In order to improve the data quality, a data processing algorithm named Kalman filter is often used to fuse data from different sources.

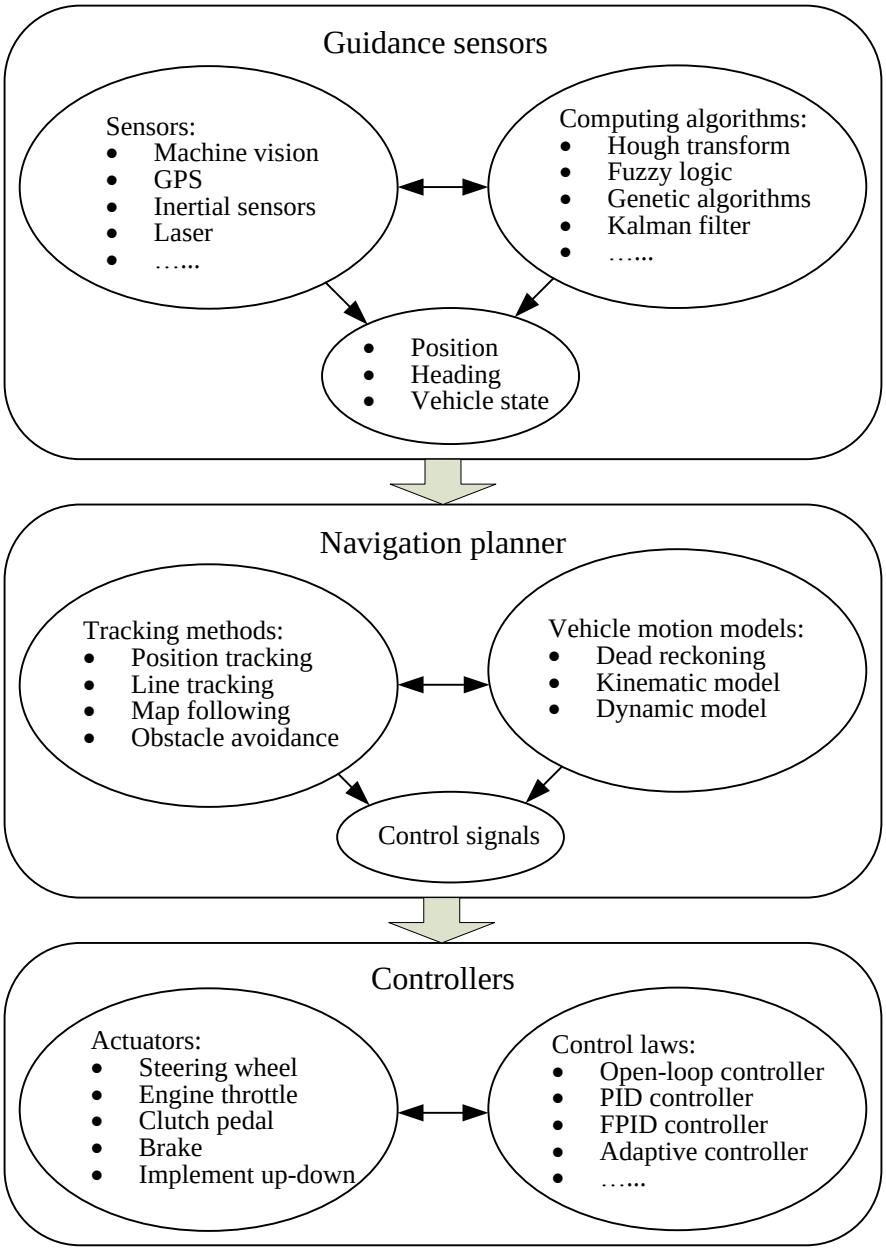


Figure 2.1: Framework of automatic guidance systems.

The second phase of an automatic guidance system is navigation planning. A navigation planner or path planner is designed to solve the problem of searching a suitable path or a series of intermediate waypoints for the vehicle to follow. In agricultural operations, local information including crop rows, swath edges, and tilled/untilled boundaries can be the source of trajectory or directrix. In parallel swathing, the trajectories are parallel lanes with an fixed offset to some prior paths. Most guidance research in agriculture has to deal with guidance in straight lanes, and thus the role of the navigation planner is much

simplified. The tracking methods or tasks for an agricultural vehicle include the following: position tracking, line tracking, map following, and obstacle avoidance.

Corrections from the navigation planner are sent to the controller for execution in real time. Proper corrections are determined based on the control algorithms and vehicle models applied in the navigation planner, and this is an important research task for autonomous vehicle development in all fields. The most common correction signal is a steering angle for the steering wheel of the guided vehicle, whereas complicated control signals for the engine throttle, clutch pedal, brake, and implement up-down can be achieved for agricultural applications (Nagasaka et al., 2004). For actuators to execute the corrections, a common method is to use calibrated step motors mounted on the steering wheel, or a sophisticated electrohydraulic steering system including electrohydraulic valves, steering hydraulic cylinders, and wheel angle measurement sensors (linear potentiometers or optical encoders) can be calibrated and mounted on the vehicle for this purpose (Rovira Más et al., 2010).

2.1.2 Applications and features

Literature about automatic guidance of agricultural vehicles using various guidance sensors is abundant. Table 2.1 lists some previous works on automatic guidance systems for agricultural applications developed around the world since the mid-1990s.

(1) GPS

A common feature of GPS-based guidance systems is that centimetre-level tracking performance can be achieved when a high-accuracy RTK-GPS is used. Researchers at Stanford University successfully developed a 4-antenna Carrier-Phase Differential Global Positioning System (CP-DGPS) for guiding a John Deere 7800 tractor on prescribed straight row paths (Bell, 2000; Bell et al., 1998; O'Connor et al., 1996). The closed-loop heading response was better than 1° and the line tracking standard deviation (SD) was better than 2.5 cm. Researchers at LASMEA used a CP-DGPS as the unique sensor for guiding a farm tractor (Thuilot et al., 2002). Field experiments showed a maximum lateral deviation of 20 cm for path tracking of a 3-m amplitude sinusoidal curve and a maximum lateral deviation of 50 cm for tracking a high-curvature curve, when the tractor was travelling at 6 km/h. Applications of RTK-GPS guidance on different platforms can be found as well, such as rice transplanters (Kurita et al., 2017; Luo & Zhang, 2007; Nagasaka et al., 2004) and tractors towing implements (Feng et al., 2005; Perez-Ruiz et al., 2012).

Table 2.1: Examples of agricultural automatic guidance systems.

Institute	Sensor	Test device	Performance	Reference
Helsinki University of Technology, Finland	DGPS, FOG	Honda TRX350	Arbitrary path, maximum lateral error 2.5 m in 567-m travel distance	(Schönberg et al., 1995)
Stanford University, USA	CP-DGPS	John Deere 7800	Heading error: bumpy field, 0.9 m/s, Mean 0.03°, SD 0.76°; Line tracking error: 50-m straight line, 0.33 m/s, Mean 1 cm, Max 10 cm, SD 2.5 cm	(O'Connor et al., 1996)
University of Illinois at Urbana-Champaign, USA	Vision sensor, RTK-GPS, FOG	Case IH Magnum 7200	3.6 m/s, maximum lateral offset less than 15 cm	(Q. Zhang et al., 1999)
LASMEA, France	CP-DGPS	Farm tractor	Sinusoidal curve: 6 km/h, lateral deviation Max 20 cm; High-curvature curve: 6 km/h, lateral deviation Max 50 cm	(Thuilot et al., 2002)
University of Illinois at Urbana-Champaign, USA	Monochrome vision sensor, DGPS	Farm tractor	Average RMS offset error 1 cm for soybean images, average RMS offset error 2.4 cm for corn images	(Han et al., 2004)
University of Illinois at Urbana-Champaign, USA	RTK-GPS, FOG	Case IH Magnum MX240	Straight or slightly curved paths, 3.5 m/s, RMS of lateral deviation less than 3 cm	(Q. Zhang & Qiu, 2004)
National Agricultural Research center, Japan	RTK-GPS, FOG	Iseki PH-6 rice transplanter	Straight path in paddy field, 0.7 m/s, deviation Max 12 cm, SD 5.5 cm	(Nagasaka et al., 2004)
University of Illinois at Urbana-Champaign, USA	Stereovision	John Deere 7700	Nearly straight crop rows, speeds 1-3 m/s, lateral error RMS 3-5 cm	(Kise et al., 2005)
South China Agricultural University, China	RTK-GPS, gyroscope, compass	SPU-60 rice transplanter	Straight path in paddy field, 0.6 m/s, lateral error Mean 14 cm, SD 25 cm	(Luo & Zhang, 2007)
University of Valladolid, Spain	RTK-GPS	John Deere 6400	Straight path: RMS of 1, 2, 3 m/s were 5.5, 5.8, 6.7 cm, respectively; Circular arc: RMS of 1, 2, 3 m/s were 17, 33, 40 cm, respectively	(Gomez-Gil et al., 2011)
Universidad de Sevilla, Spain	RTK-GPS	John Deere 6430 tractor towing transplanter	Straight row: 1.6 km/h, lateral error Mean 3.22 cm, SD 1.58 cm; Curved row: 1.6 km/h, lateral error Mean 2.68 cm, SD 1.34 cm	(Perez-Ruiz et al., 2012)
Middle East Technical University, Turkey	FOG, LRF	Toro Workman MDE utility vehicle	Seven straight rows of total length 370 m, lateral errors mostly concentrated around zero, maximum deviations around 20 cm	(Bayar et al., 2015)
Ritsumeikan University, Japan	Vision sensor, GPS, compass	Rice combine harvester VY446LM	Straight paths in paddy field, lateral error Max 0.12 m, RMS 0.04 m, azimuth error Max 8.4°, RMS 2.6°	(Kurita et al., 2017)

(2) Machine vision

Vision systems can guide a vehicle to track paths within errors of a few centimetres as well when environment conditions are good enough. Benson et al. (2003) developed a machine vision guidance system based on a Cohu 2100 monochrome camera for a Case 2188 combine harvester. Experimental tests of the system for corn harvesting showed the

same level accuracy as a GPS-based guidance system. B. Chen et al. (2003) developed a vision system based on a Sony DCR-PC10 camera for a rice transplanter. The system was capable of analyzing images of field shorelines and rows of seedlings as well as detecting the row and field ends. Rovira Más et al. (2004) used a MEGA-D stereo camera to find paths from structured agricultural fields to automatically navigate a tractor. Field tests showed that the deviations from the reference path were less than 5 cm when the tractor was autonomously travelling at 2.7 m/s. A stereovision-based crop row detection method for a John Deere 7700 tractor was developed by Kise et al. (2005). Field validation tests indicated that the automatic guidance system could localize crop rows accurately and reliably in a weedy field with missing sections of soy beans. The root mean square (RMS) of the cross-track error of nearly straight rows was 3 to 5 cm when the tractor was travelling at speeds of 1 to 3 m/s. The vineyard equipment manufacturer Clemens GmbH & Co used a three-dimensional (3D) camera system to guide a tractor along the narrow tracks of vineyards. An accuracy of ±3 cm was obtained and the system could cope with vegetation variations over the seasons (Möller, 2010). Xue et al. (2012) developed a variable field-of-view machine vision method to navigate an agricultural robot in cornfields. The main guidance sensor was a low-cost digital Logitech camera. Field tests showed a maximum lateral error of 15.8 mm for travelling a distance of 30 m.

(3) Laser

Guidance systems using laser sensors can also be found, although they are not as common as systems using GPS and vision sensors. Chateau et al. (2000) developed an automatic guidance system using a one-dimensional scanning LRF for an agricultural harvester working in a structured environment. Ahamed et al. (2004, 2009) used a SICK laser measurement system (LMS) 211 LRF for infield navigation and farm implements auto-hitching. Tsubota et al. (2004) developed a laser scanner based guidance system for a John Deere Gator tractor working in an orchard. When the vehicle travelled along a straight line at 1 m/s, the RMS of the lateral and directional errors were 10 cm and 0.7° respectively. Hiremath et al. (2014) developed a probabilistic sensor model based on a two-dimensional (2D) LRF to guide a small agricultural robot. Fields tests showed that the RMS of the heading and lateral deviations were 2.4° and 0.04 m respectively.

(4) Sensor fusion

The integration of different guidance sensors to obtain a robust, reliable, and adaptive navigation solution is actively researched. Nagasaka et al. (2002, 2004) developed an

automated operation system for a rice transplanter based on an RTK-GPS and fibre optic gyroscope (FOG) sensors. The FOG sensors were used to measure the direction and inclination of the machine. Lateral deviations of less than 10 cm from the field tests were observed when the travel speed of the transplanter was 0.8 m/s. Further development which integrated the RTK-GPS with a high-grade IMU was conducted (Nagasaka et al., 2009). Field tests showed that the RMS and mean value of the lateral deviations were less than 4 cm and 3 cm respectively, and the RMS and mean value of the heading errors were less than 3.6° and 3.4° respectively. Guo et al. (2003) integrated a low-cost Garmin GPS and a set of solid-state inertial sensors to develop a positioning system for a John Deere Gator tractor. Inertial sensors used in the system included three single-axis gyros and one triaxial accelerometer. Sensor data fusion was realized by a position-velocity-attitude model based Kalman filter. Results of experimental tests on both field paths and paved paths indicated that the low accuracy of the original GPS (3 m) could be improved to sub-metre level (0.3-0.5 m). A guidance system based on the integration of an RTK-GPS, a micro-electromechanical system (MEMS) gyro, and an electronic compass was developed for a Kubota SPU-60 rice transplanter (Luo & Zhang, 2007). Experimental tests on paddy fields showed that at a travel speed of 0.6 m/s, the path tracking errors were less than 0.15 m and the headland turning radii were less than 1 m.

Developments on the combination of machine vision technology with other sensors can be found as well. Q. Zhang et al. (1999) developed a navigation system with redundant sensors including a CCD camera, a FOG, and an RTK-GPS for a Case IH Magnum 7200 tractor. Field tests of sensor fusion, vision only, and GPS+FOG only guidance were conducted and compared. When the travel speed was set at 3.6 m/s, the maximum offsets of the vehicle to a desired path were less than 0.15 m with sensor fusion, 0.20 m with GPS+FOG, and 0.15 m with vision only based steering control strategies. The fusion-based navigation could provide satisfactory vehicle guidance when individual navigation signals, such as images of the path or satellite signals, were lost for short periods of time.

Sensor fusion is a method used to incorporate multiple sensing sources to obtain more comprehensive, reliable information of the controlled vehicle. Kalman filter is a sensor fusion technique that has been widely studied and applied since its invention in the 1960s. Geng et al. (2007) used two Kalman filters for robust estimation of navigation parameters and errors of inertial sensors. The first one was an adaptive fading factor Kalman filter, and a GPS dynamic model was used to generate the velocities and accelerations that were further used to acquire approximate pitch and heading values. The second filter was used to integrate the position, velocity, and attitude from the IMU and the GPS. Researchers at the University of New South Wales developed a fusion algorithm AhrsKf (Attitude Heading Reference System Kalman Filter) for an automated agricultural vehicle (Cole,

2008; Y. Li et al., 2012). Differing from the complementary filtering approach, the AhrsKf used a loosely coupled integration Kalman filter to fuse data from the GPS and MEMS inertial sensors. Researchers at the University of Florida developed a fuzzy logic enhanced Kalman filter to fuse information from a vision camera, a laser radar, an IMU, and a speed sensor (Subramanian et al., 2009). The fused information was used to guide a vehicle running in citrus groves. Experimental tests showed that sensor fusion based guidance was more accurate than guidance using independent sensors.

2.1.3 Related works

The majority of research activities in agricultural automatic guidance dealt with single-vehicle platforms and continuous path following of straight lines or curves. There exist a few studies about point approaching and multi-vehicle cooperation tasks which represent some similarities to the work in this thesis.

Researchers at the University of Tsukuba developed a control method for a front-wheel steering tractor to approach target points to perform automatic fertilizer refilling (Sutiarso et al., 2002; Takigawa et al., 2002). The control input for the steering angle was designed based on a combination of polynomial and cosine functions, and the trajectory for the tractor was generated in polar coordinates. A FOG and rotary encoders were installed on the tractor for positioning. Since the experiments were run for one single point approach in a short time period, the drift error of the gyroscope was negligible. Experimental tests showed that positional errors varied from 2.5 to 39 cm when the targets were placed with distances of 4 to 12 m to the start points. The maximum error occurred when the theoretical control inputs exceeded the saturation range, which was -40° to 40°.

Researchers at the University of Hokkaido developed a master-slave robot tractor system for farm operations (Noguchi et al., 2004). Two motion control algorithms, namely a GOTO algorithm and a FOLLOW algorithm, were developed (Figure 2.2). The GOTO algorithm could be applied when the slave robot was asked to go to a specific place, a certain distance from the current operation position. The FOLLOW algorithm allowed a more cooperative way to guide the slave to follow the master at a predetermined relative distance and angle, regardless of the travel speed and direction. A nonlinear sliding mode controller for the slave was used to provide robust control, and validation tests indicated that the sliding mode controller had better performance on both lateral offset control and spacing control than a conventional proportional-derivative controller. The two tractors were equipped with RTK-GPS receivers and gyroscopes for positioning, and the communication between them was realized through a wireless local area network at 2 Hz.

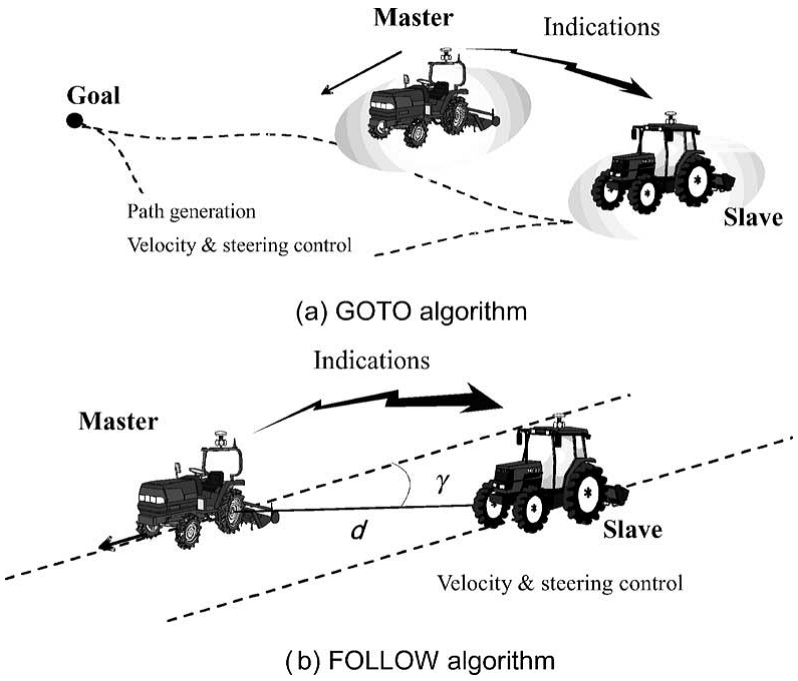


Figure 2.2: A master-slave robot system, (a) GOTO algorithm; (b) FOLLOW algorithm (Noguchi et al., 2004).

Researchers at the University of Florida tested a forward movement synchronization control system for two tractors working in parallel (Lee et al., 2007). The system used a laser scanner LMS 200 to detect the relative positions of the two tractors, as shown in Figure 2.3. The control system was mounted on the slave tractor to synchronize its forward movement with the master tractor's. A plastic pipe was mounted vertically on the master as a target for the laser scanner on the slave. Stationary tests showed that distance errors using the laser scanner were between $\pm 0.31\%$ and $\pm 0.37\%$ for lateral distance ranges of 5.2 to 6.8 m between the two vehicles (SD$<$7 mm). Dynamic tests using constant travel speeds showed that the maximum lead error and maximum lag error were 32 cm and 23 cm respectively. Dynamic tests using varying speeds with a trapezoidal change pattern (from 0.64 to 1.14 m/s) showed 17.3 cm and 10.9 cm for the maximum lead and maximum lag respectively.

Researchers at the Institute of Karlsruhe Technology developed a semi-autonomous master-slave system between two agricultural vehicles, as shown in Figure 2.4 (X. Zhang et al., 2010). The driverless slave tractor was controlled to follow the manually-operated master with given lateral and longitudinal offsets. The reference course for the slave to follow was generated in three modes: standard, obstacle avoidance, and turning. An RTK-GPS was used for the positioning of the tractors and control signals were exchanged through XBee wireless modules.

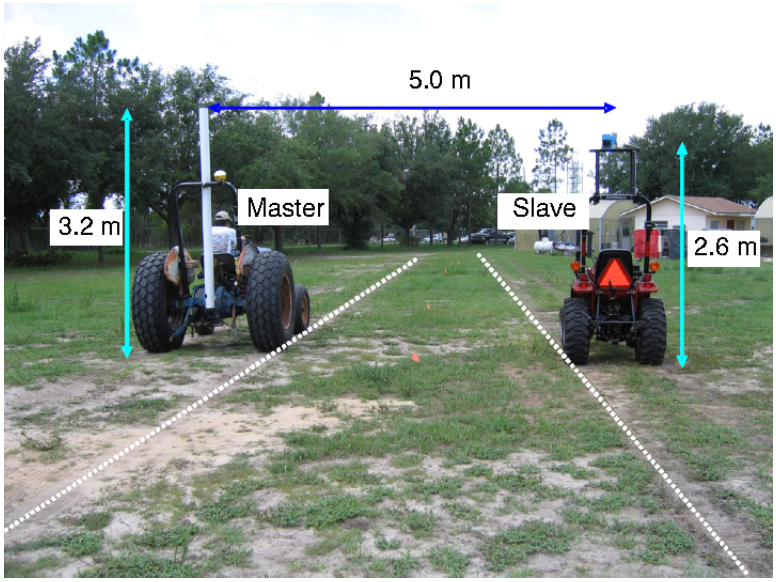


Figure 2.3: A master-slave forward movement synchronization system using a laser scanner (Lee et al., 2007).

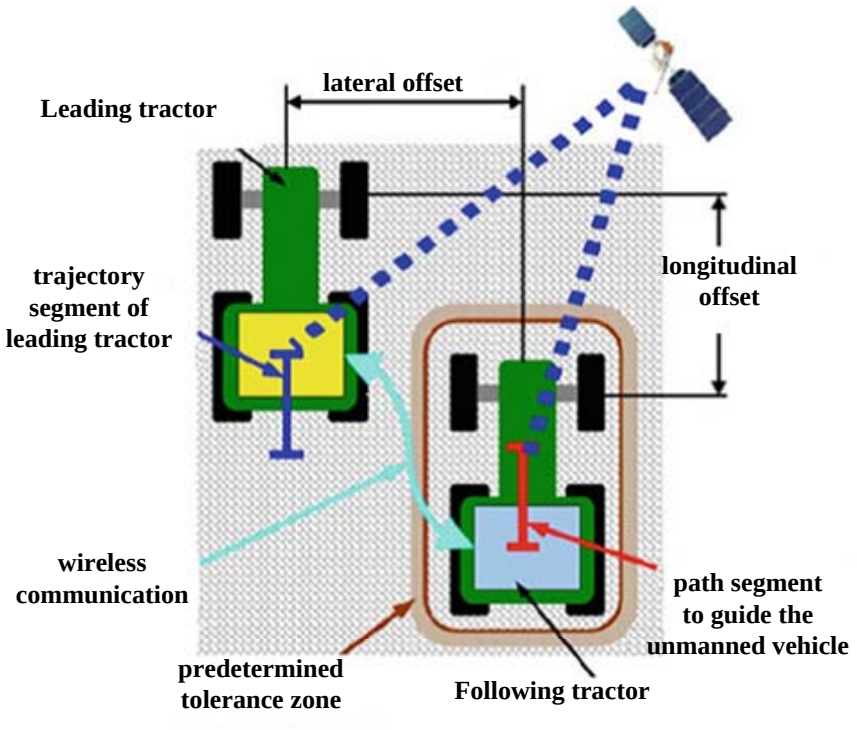


Figure 2.4: A master-slave vehicle system using RTK-GPS and wireless communication (X. Zhang et al., 2010).

A leader-follower system using two robot tractors was developed by researchers at the University of Hokkaido (C. Zhang et al., 2016). The two robot tractors could work together to form a certain spatial arrangement during the operation and make headland turns without collision. On each tractor, an RTK-GPS, an IMU, a laser scanner, an onboard PC, and a Bluetooth unit were used. The position and motion data were provided by the GPS and IMU sensors. The laser scanner was used for collision avoidance and the communication between the two robot tractors was established through Bluetooth units wirelessly. Field experiments showed an average lateral error of less than 4 cm and an improved work efficiency of 95.1%.

2.1.4 Summary

GPS-based and vision-based systems are the two main types among guidance systems applied on agricultural machines. There appear to be two main limitations to the use of GPS guidance. First is that the GPS cannot be used in microwave-shielded areas and it encounters satellite signal blockage or multi-path errors when the system is used in presence of high-rise buildings, trees or steeply rolling terrain. The second limitation is the inherent time delay required for signal processing to determine locations, which might present challenges at high field speeds. However, signal outage should not be encountered when the vehicles run on wide open farms and multi-path errors can be reduced to minimum amounts after an appropriate arrangement of the equipment. The existing GPS products usually provide an update rate of 5, 10, and up to 20 Hz. This can meet the velocity requirements for most infield agricultural operations. The most accurate RTK-GPS provides positioning information with a few centimetres accuracy, but the cost is much higher than mid-range accuracy GPS products. Although it has been shown that the accuracy of a low-cost DGPS can be improved to sub-metre level by integrating it with inertial sensors and applying sensor fusion techniques, this accuracy level is still not applicable to some critical operations such as planting and harvesting.

Vision systems are especially suitable when well-constructed information is available, e.g., clear field boundaries, uniform crop rows. Algorithms for processing images and extracting the guidance information are usually complex and computationally expensive. This restricts the application of vision systems to high-speed operations. However, with the advent of faster and faster processors, this will no longer be a challenge. Another limitation of vision systems is that they are sensitive to environmental variables. Weeds in the field, ambient lights, and tree shades all negatively affect their performances. Applying a well-built vision guidance system for a particular crop or field to different crops or field conditions may not be easy. These facts impede the commercialization of vision-based

guidance systems.

Laser sensors can achieve high positioning accuracy on condition that suitable reflectors exist, either naturally or artificially. Laser-based guidance systems can be used in orchards where GPS signals are blocked or refracted. Applying laser systems for guidance in open fields needs the setup of artificial reflectors or beacons, which takes up time and labour and is obviously not applicable to large fields.

Currently, there exists virtually no research on automatic guidance of large agricultural platforms having spans longer than 30 m and the number of research on vehicle-vehicle cooperation is limited. However, some valuable suggestions can still be extracted. The synchronous motion control of two vehicles in parallel using a laser scanner is feasible when they are close to each other. Because the detecting accuracy of a laser scanner degrades as the scanning range extends, it is not reliable to use a laser sensor for real-time vehicle-vehicle cooperation when they are far apart. The combination of RTK-GPS with wireless communications tends to be a suitable method for autonomous motion control of multiple-vehicle cooperation. The high-accuracy RTK-GPS permits that the positioning requirements of the vehicles are met, and the high data rate and long application range of the wireless modules allow real-time control of a group of vehicles on a wide coverage area.

2.2 Motion control of mobile robots and vehicles

2.2.1 Nonholonomic systems and control tasks

Motion control problems can be classified into two main categories: holonomic and nonholonomic. On a 2D plane, when the three degrees of freedom of a wheeled mobile robot or vehicle can be manipulated independently, the system is called a holonomic one. A mobile robot with omni wheels belongs to this category as it can roll both forward and sideways. The nonholonomic nature of ordinary mobile robots and car-like vehicles is related to the assumption that wheels roll without slipping. This can be seen from the fact that cars cannot move to the left or right without changing its orientation. From a control point of view, nonholonomic systems are characterized by non-integrable first-order differential constraints on the configuration variables. A simple example is a unicycle rolling freely on a 2D surface, with x, y and θ defining its configuration, as shown in Figure 2.5. The kinematic motion equation of the unicycle can be written as

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{2.1}$$

with v and ω being the input linear and angular velocities. The configuration of the unicycle is constrained by the following differential equation

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{2.2}$$

which cannot be integrated to form a relationship between x, y and θ.

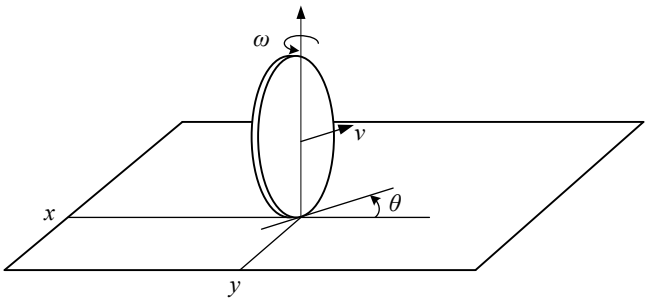


Figure 2.5: A unicycle rolling on a plane.

Wheeled mobile robots and autonomous car-like vehicles have wide applications in areas including military, transportation, manufacturing, and agriculture. Extensive research has been conducted on nonholonomic mobile robots completing various motion tasks since the late 1980s. These tasks can be classified into three basic problems: tracking a reference trajectory, following a reference path, and stabilizing to a desired point/posture. The three tasks are sketched in Figure 2.6, with reference to a car-like vehicle.

- **Point stabilization**: a problem that is concerned with the design of control inputs that force a vehicle to reach and stop at a desired goal configuration starting from a given initial configuration.
- **Path following**: a problem that is concerned with the design of control inputs that force a vehicle to reach and follow a geometric reference with time-free parametrization.
- **Trajectory tracking**: a problem that is concerned with the design of control inputs that force a vehicle to reach and follow a geometric reference with an associated timing law.

The nonholonomic constraints reduce the instantaneous motions that the robots and vehicles can perform to accomplish the desired motion tasks. Nevertheless, the global controllability in the configuration space can still be achieved. It is noted that the point

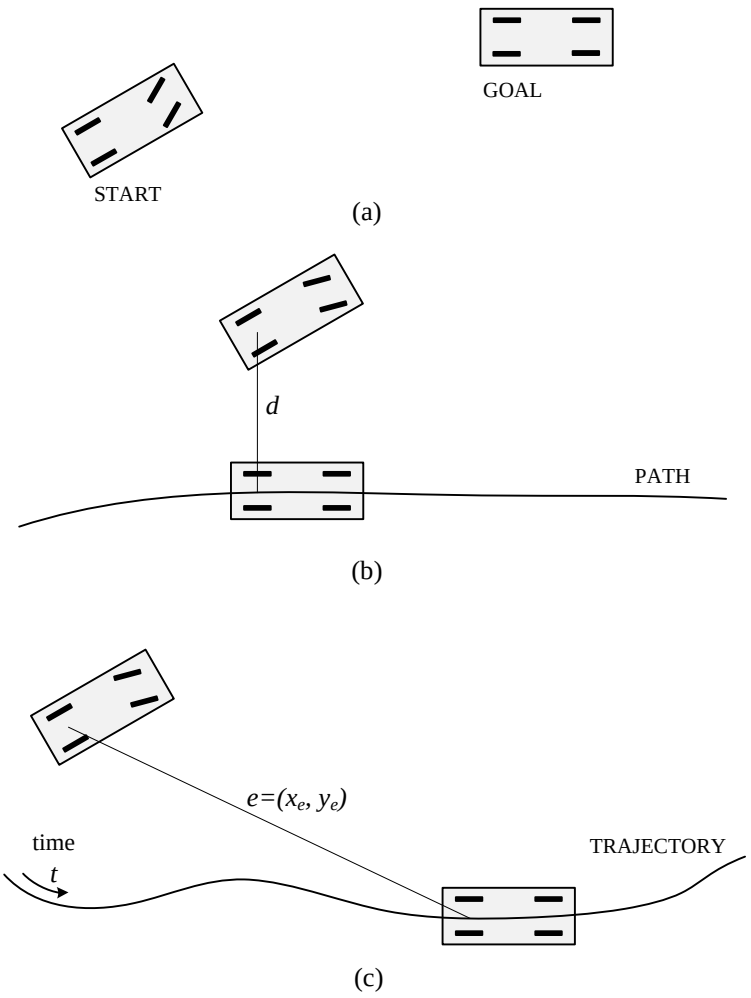


Figure 2.6: Motion tasks of a car-like vehicle, (a) point stabilization; (b) path following; (c) trajectory tracking (De Luca et al., 1998).

stabilization or point-to-point motion task is more difficult than path following and trajectory tracking. As is implied by Brockett's theorem, it is not possible to stabilize the system of point-to-point motion by using smooth time-invariant feedback control laws (Brockett, 1983). The difference between point stabilization and path following/trajectory tracking can be explained by drawing a comparison between the number of inputs and the number of outputs to be controlled. For a case of a unicycle-like robot, the point stabilization task is actually an input-state problem with two velocity inputs, v and ω, and three controlled states, namely x, y and θ. The path following task is an input-output problem with one velocity input ω and one controlled output (the distance d to the path in Figure 2.6(b)), and the trajectory tracking task is again an input-output problem with two velocity inputs (v and ω) and two controlled outputs (the errors x_e and y_e to the reference vehicle in Figure 2.6(c)). The path following and trajectory tracking tasks have a similar

level of difficulty, being "square" control (same number of control inputs and controlled outputs) (De Luca et al., 1998).

2.2.2 Vehicle control models

(1) Dead reckoning

Dead reckoning (DR) is a relative positioning method which calculates the vehicle's current position by advancing the known initial position based upon the integration of the vehicle's velocity and heading direction over a given length of time. The simplest form of DR is often termed as odometry, which implies that the vehicle's displacement along the travel path is directly derived from some onboard "odometer". DR is reliable for short distance measurements on a smooth uniform surface. The major problem of DR is its unbounded accumulated error, which derives from the integration of sensor measurements over time. At an earlier stage, DR was used independently to measure a vehicle's motion for position information (Freeland, 1990; Freeland et al., 1989). Nowadays, DR is widely used as an auxiliary method to some main guidance methods such as GPS and machine vision.

(2) Kinematic models

Kinematic models describe the motion of wheeled mobile robots and vehicles by a set of first-order differential equations with regard to its configuration variables and control inputs. For a unicycle-like or differentially-driven mobile robot, the kinematic model is described by equation (2.1). When it comes to a car-like vehicle, the kinematic model equation becomes

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = v\frac{\tan\delta}{l} \end{cases} \tag{2.3}$$

where δ is the front-wheel steering angle and l is the vehicle wheelbase. The main feature of the kinematic model is the presence of nonholonomic constraints due to the rolling without slipping condition between the wheels and the ground. This condition is generally satisfied and the kinematic model is applicable when the vehicle is operated at slow speeds and the wheel/ground contact surface is level and firm. Model equations (2.1) and (2.3) are the basic descriptions of vehicle motions at the kinematic level. For control design purposes, the kinematic model is often described as or derived into other forms depending on the particular motion task.

For a path following problem depicted in Figure 2.7, the kinematic model is derived in terms of path coordinates as follows (De Luca et al., 1998; Thuilot et al., 2002):

$$\begin{cases} \dot{s} = \frac{v\cos\theta_p}{1-dc(s)} \\ \dot{d} = v\sin\theta_p \\ \dot{\theta}_p = v\left(\frac{\tan\delta}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right) \end{cases} \tag{2.4}$$

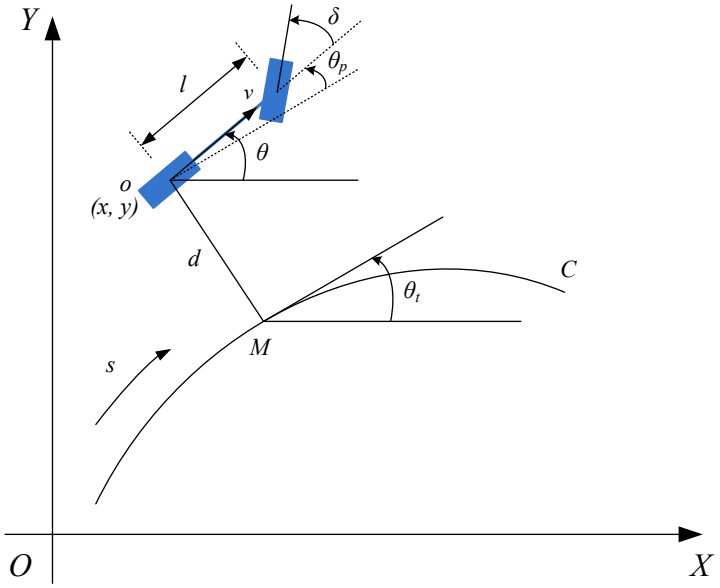


Figure 2.7: Kinematic model for a path following task.

The relevant variables appearing in the model are denoted as follows:

- C: the path for the vehicle to follow,
- M: the reference point on C which is the closest to the centre of the vehicle virtual rear wheel o,
- $(x,\, y,\, \theta)$: the coordinates of the vehicle o with respect to the inertial frame (OXY),
- s: the curvilinear coordinate of point M along C, and $c(s)$ denotes the curvature of C at that point,
- d: the lateral error of the vehicle to the reference point M,
- θ_t: the tangent of the path at point M with respect to the X-axis,
- θ_p: the orientation error of the vehicle with respect to the path C, $\theta_p = \theta - \theta_t$,
- δ: the steering angle of the virtual front wheel,
- v: the linear velocity of the vehicle with respect to the inertial frame,
- l: the vehicle wheelbase.

For a trajectory tracking problem depicted in Figure 2.8, a reference vehicle is assumed on the trajectory and three tracking errors are defined with the geometric relationship as

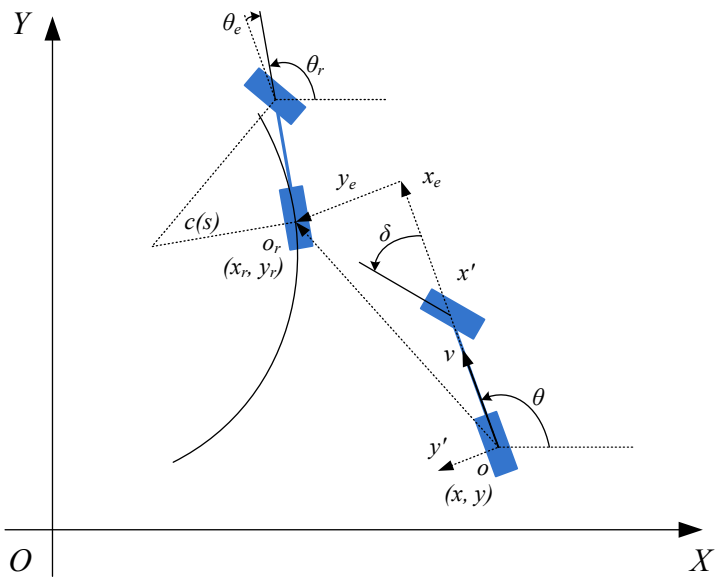


Figure 2.8: Kinematic model for a trajectory tracking task.

$$\begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{pmatrix} \tag{2.5}$$

Differentiating the above equation leads to the following kinematic error model (Fang et al., 2006; Kanayama et al., 1991):

$$\begin{cases} \dot{x}_e = -v + v_r \cos\theta_e + \omega y_e \\ \dot{y}_e = v_r \sin\theta_e - \omega x_e \\ \dot{\theta}_e = v_r c(s) - \frac{v}{l}\tan\delta \end{cases} \tag{2.6}$$

The relevant variables appearing in the model are denoted as follows:

- $(o,\ x',\ y')$: the vehicle body frame, and o is the point to be explicitly controlled,
- $(x,\ y,\ \theta)$: the coordinates of the vehicle o with respect to the inertial frame (OXY),
- $(x_r,\ y_r,\ \theta_r)$: the coordinates of the reference vehicle o_r with respect to the inertial frame,
- $(x_e,\ y_e)$: the tracking errors depicted in the vehicle body frame $(ox'y')$,
- θ_e: the orientation error, $\theta_e = \theta_r - \theta$,
- s: the curvilinear coordinate of the reference point o_r, and $c(s)$ denotes the curvature

of the reference path at that point,

- δ: the steering angle of the virtual front wheel,
- $v(v_r)$: the linear velocity of the (reference) vehicle with respect to the inertial frame,
- $\omega(\omega_r)$: the angular velocity of the (reference) vehicle with respect to the inertial frame, $\omega_r = v_r c(s)$,
- l: the vehicle wheelbase.

For a point stabilization problem, there exits no smooth time-invariant control law because the system model violates Brockett's necessary condition for smooth stabilizability (Brockett, 1983). To avoid this deficit, the vehicle model can be transformed and represented in terms of its polar coordinates. As first introduced by Aicardi et al. (1994, 1995), after the Cartesian to polar coordinates transformation, a different form of the kinematic model can be derived and simple Lyapunov design methods can be applied. The model is depicted in Figure 2.9 with the relevant variables denoted as follows:

- (x, y, θ): the Cartesian coordinates of the vehicle o with respect to the inertial frame (OXY),
- (x_d, y_d, θ_d): the Cartesian coordinates of the vehicle o with respect to the inertial frame (OXY) at the desired point,
- δ: the steering angle of the virtual front wheel,
- ρ: the linear distance error between the current vehicle position and the target point,
- α: the heading error, which is the angle of the distance line with respect to the vehicle velocity line,
- β: the orientation error, which is the angle of the distance line with respect to the desired orientation,
- ψ: the angle of the distance line with respect to the X-axis,
- l: the vehicle wheelbase.

The coordinates transformation is done through the following equation:

$$\begin{cases} \rho = \sqrt{(x - x_d)^2 + (y - y_d)^2} = \sqrt{\Delta x^2 + \Delta y^2} \\ \psi = \text{ATAN2}\left(\Delta y, \Delta x\right) \\ \alpha = \psi - \theta \\ \beta = \psi - \theta_d \end{cases} \tag{2.7}$$

and the kinematic model with transformed state variables is derived as follows:

$$\begin{cases} \dot{\rho} = -v \cos\alpha \\ \dot{\alpha} = -v \left(\frac{\tan\delta}{l} - \frac{\sin\alpha}{\rho} \right) \\ \dot{\beta} = \frac{v \sin\alpha}{\rho} \end{cases} \tag{2.8}$$

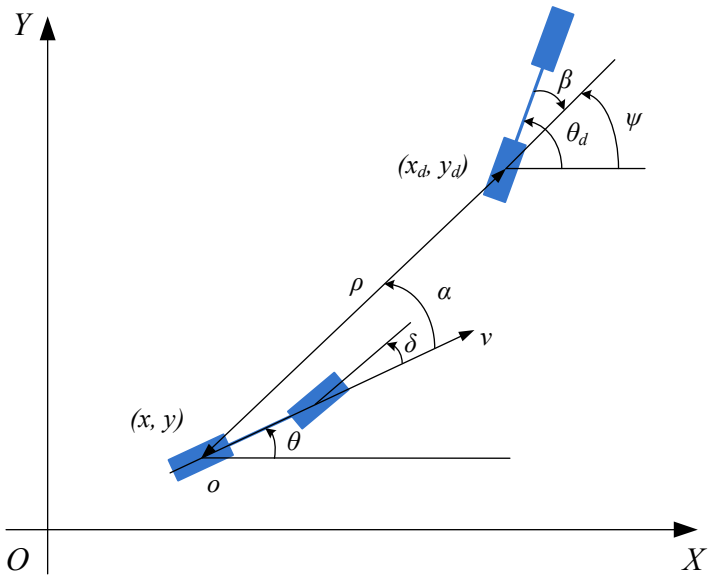


Figure 2.9: Kinematic model for a point stabilization task.

(3) Extended kinematic models

The ideal kinematic models of mobile robots and vehicles are derived based on pure rolling constraints. These constraints are generally satisfied for robots operating under controlled conditions, e.g., indoor floors, hard paved roads, and slow speeds. However, due to various dynamic effects such as tire deformation and side slipping, the pure rolling conditions are never strictly satisfied, especially when vehicles are running off-road under high speeds. In agricultural applications, farm vehicles are required to move on all-terrain grounds such as slippery slopes and muddy grass grounds where contact conditions between tires and grounds are unpredictable. Sliding and skidding inevitably occur in these situations, and the performance of automatic guidance relying on ideal kinematic models is deteriorated. To incorporate the possible sliding and skidding effects, the ideal kinematic models are refined by adding additive parameters and extended kinematic models are obtained. Although the sliding and skidding effects are related to the vehicle dynamics and exact descriptions of the effects rely on vehicle dynamic models, those additive parameters describe, to a large extent, the correct vehicle behaviours when sliding or skidding occurs.

A series of research work was conducted on using extended kinematic models for automatic guidance of agricultural vehicles. Lenain et al. (2003) introduced two additional parameters, a linear lateral velocity $\dot{Y}_p$ and an angular velocity $\dot{\Theta}_p$, into the path following model (equation (2.4)) to account for the sliding phenomena. The kinematic model was

refined as (the first equation in (2.4) was dropped)

$$\begin{cases} \dot{d} = v\sin\theta_p + \dot{Y}_p \\ \dot{\theta}_p = v\left(\frac{\tan\delta}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right) + \dot{\Theta}_p \end{cases} \tag{2.9}$$

Fang et al. (2005) derived a different model form, in which a lateral velocity v_y and a steering angle bias δ_b were introduced. The model equation was refined as

$$\begin{cases} \dot{s} = \frac{v\cos\theta_p}{1-dc(s)} - \frac{v_y\sin\theta_p}{1-dc(s)} \\ \dot{d} = v\sin\theta_p + v_y\cos\theta_p \\ \dot{\theta}_p = v\left(\frac{\tan(\delta+\delta_b)}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right) - \frac{v_y}{l} + \frac{c(s)v_y\sin\theta_p}{1-dc(s)} \end{cases} \tag{2.10}$$

A similar model was derived by Lenain et al. (2006), in which two parameters, a front side slip angle β_p^F and a rear side slip angle β_p^R, were introduced to describe the sliding behaviours. The model equation was refined as

$$\begin{cases} \dot{s} = \frac{v\cos\left(\theta_p+\beta_p^R\right)}{1-dc(s)} \\ \dot{d} = v\sin\left(\theta_p+\beta_p^R\right) \\ \dot{\theta}_p = v\left(\cos\beta_p^R\frac{\tan\left(\delta+\beta_p^F\right)-\tan\beta_p^R}{l} - \frac{c(s)\cos\left(\theta_p+\beta_p^R\right)}{1-dc(s)}\right) \end{cases} \tag{2.11}$$

For a farm vehicle to accomplish a trajectory tracking task, Fang et al. (2006) derived an extended kinematic model based on the ideal model (equation (2.6)). The sliding effects were described by three additive parameters: the longitudinal sliding velocity v_x^s, the lateral sliding velocity v_y^s, and the bias of the steering angle δ_b. The model equation was refined as

$$\begin{cases} \dot{x}_e = -\left(v_\omega + v_x^s\right) + v_r\cos\theta_e + \omega y_e \\ \dot{y}_e = -v_y^s + v_r\sin\theta_e - \omega x_e \\ \dot{\theta}_e = v_r c(s) - \left(\frac{(v_\omega+v_x^s)}{l}\tan\left(\delta+\delta_b\right) - \frac{v_y^s}{l}\right) \end{cases} \tag{2.12}$$

with v_ω being the rear-wheel rotating velocity. Longitudinal sliding was considered in this model and it was compensated in order to realize trajectory tracking. Low & Wang (2008a,b) developed two similar extended kinematic models to equations (2.11) and (2.12) for a car-like mobile robot to accomplish path following and trajectory tracking tasks. In their developed models, the skidding and slipping effects of the wheels were characterized by front and rear slipping angles.

(4) Dynamic models

In kinematic models, the linear velocity v and angular velocity ω (or steering angle δ for a car-like robot) are deemed as control inputs directly. However, in reality, these control inputs cannot be achieved instantaneously. For example, the physical inputs for generating the linear velocity of a mobile robot are the electric motor torques or input voltages. Perfect tracking of control inputs for the kinematic models is not realistic because of hardware limitations. In the work of Thuilot et al. (2002) and Lenain et al. (2006), the electrohydraulic steering mechanism of a farm tractor was identified as a second-order process with a pure delay. For kinematic models to work properly, a two-level control scheme is usually applied. At the upper level, desired steering angles and velocities are calculated based on control laws designed from the kinematic models. The lower level usually applies a feedback control loop to keep the hardware outputs tracking the desired values consistently.

Dynamic models are required to fully describe the behaviours of mobile robots and vehicles. As velocity terms are not controlled directly, dynamic models are considered and control laws are designed on the torque level for motor-drive wheeled mobile robots. The well-known dynamic equation of a robot system with n generalized coordinates and m nonholonomic constraints is given as (Sarkar et al., 1994)

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)\tau - A^T(q)\lambda \tag{2.13}$$

where M is a symmetric positive definite inertia matrix, V_m is the centripetal and Coriolis matrix, F is the surface friction vector, G is the gravitational vector, τ_d denotes bounded unknown disturbances including unstructured modelled dynamics, B is the input transformation matrix, τ is a control input vector, A is a matrix associated with the nonholonomic constraints, λ is a vector of constraint forces, and $\dot{q}$ and $\ddot{q}$ denote the velocity and acceleration vectors respectively. For small-scale experimental mobile robots, the dynamic parameters, including the mass, the moment of inertia about the gravity centre, the wheel stiffness, and geometrical parameters (the wheelbase and the radius of the wheels), can be conveniently determined. Example applications of using dynamic model (2.13) for mobile robot motion control can be found in Fierro & Lewis (1997a), Mnif (2004), C.-Y. Chen et al. (2009), and Jang (2011).

The dynamic description of a farm vehicle is fairly complex and describing all vehicle features including inertia, sliding, and springing leads to large, intricate models. A detailed analysis of tractor dynamics can be found in Rovira Más et al. (2010, Chapter 2), in which the lateral dynamics and the steering system were modelled based on a bicycle model. Pa-

rameters including the vehicle mass, the turning inertia with respect to the vehicle gravity centre, and the cornering stiffnesses of the front and rear tires were needed in the model. In most agricultural guidance applications, kinematic models were considered because accurate values of the dynamic parameters were difficult to determine in an off-road context, and some parameters such as the cornering stiffness can vary with changes in velocity, axle load, and the ground surface. In Q. Zhang & Qiu (2004), the dynamic model of a farm tractor was studied and a path search algorithm for automatic navigation was developed. The side forces acting on the wheels were described by functions of wheel slip angles and cornering stiffnesses. The dynamic parameters were identified based on a series of dynamic response tests. For off-road vehicles moving under complex environments, sliding occurs inevitably and the full description of the sliding effects relies on dynamic models of the vehicles. Since the tire cornering stiffness is one of the most important parameters for lateral stability, estimation of the cornering stiffness plays a key role in sliding estimation and compensation. A dynamic observer based approach for online estimation of the sliding effects for off-road vehicles was applied in Lenain et al. (2009) and Fang et al. (2011).

2.2.3 Motion control methods

Various control methods have been applied to solve the three basic motion tasks of mobile robots. Brockett's theorem (Brockett, 1983) proved that nonholonomic systems cannot be asymptotically stabilized around an equilibrium using smooth time-invariant feedback control laws. Hence, many researchers endeavoured to find novel control methods to solve the stabilization problem of nonholonomic mobile robots. Literature about motion control of mobile robots is abundant. Table 2.2 lists some representative works, of which some were significant theoretical developments and some extended to experimental validations.

Table 2.2: Motion control methods of mobile robots.

Task	Control method	Characteristics	Validation	Reference
Point stabilization	Piecewise sinusoids in chained form	Open-loop control, piecewise trajectories	no	(Murray & Sastry, 1990, 1991, 1993)
Trajectory tracking	Lyapunov function	Error reference model	yes	(Kanayama et al., 1991)
Point stabilization	Piecewise smooth control	Infinite switching avoided	no	(De Wit & Sørdalen, 1992)
Point stabilization	Lyapunov function in polar coordinates	Continuous time-invariant control laws	no	(Aicardi et al., 1994, 1995)
Path following, point stabilization	Lyapunov function on skew-symmetric chain form	Globally stabilizing smooth time-varying control laws	no	(Samson, 1995)

Continued on next page

Table 2.2 – *Continued from previous page*

Task	Control method	Characteristics	Validation	Reference
Trajectory tracking	Integrator backstepping	From velocity to torque control	no	(Fierro & Lewis, 1995)
Trajectory tracking	Static and dynamic feedback linearization	Switching to time-varying control to avoid singularities	no	(D'Andréa-Novel et al., 1995)
Path following	Partial state feedback linearization	Globally asymptotically stable	yes	(Tayebi & Rachid, 1996)
Trajectory tracking, point stabilization	Backstepping control and neural networks	Unmodelled bounded disturbances and unstructured unmodelled dynamics	no	(Fierro & Lewis, 1997a,b)
Trajectory tracking	Time-varying feedback with integrator backstepping	Local and global tracking with exponential convergence	no	(Jiang & Nijmeijer, 1997)
Point stabilization	Discontinuous time-invariant state feedback in polar coordinates	Velocity/torque control with model errors and noise	no	(Astolfi, 1999)
Trajectory tracking	Sliding mode control in polar coordinates	Presence of bounded external disturbances with geometric constrains	yes	(Yang & Kim, 1999)
Point stabilization	Lyapunov function in polar coordinates	Forward motion only	no	(Indiveri, 1999)
Point stabilization	State feedback linearization in polar coordinates	Linear time-invariant system transformation	no	(Park et al., 2000)
Trajectory tracking	Adaptive control with backstepping	Dynamic model with unknown parameters	no	(Fukao et al., 2000)
Trajectory tracking	Sliding mode control	Presence of parameter variations and input disturbances	no	(Corradini & Orlando, 2001)
Trajectory tracking, point stabilization	Adaptive control with backstepping	Unknown dynamic parameters	no	(Pourboghrat & Karlsson, 2002)
Point stabilization	Discontinuous state feedback with backstepping in chained form	Discontinuous control for car-like mobile robots	no	(Mnif, 2004)
Trajectory tracking, point stabilization	Sliding mode control in polar coordinates	Torque control with input disturbances	no	(Chwa, 2004)
Point stabilization	Four-posture control in polar coordinates	Control with four possible moving regions	yes	(Shim & Sung, 2004)
Trajectory tracking	Fuzzy logic control based on Kanayama et al. (1991)	Consecutive waypoints and look-ahead curvature	yes	(Maalouf et al., 2006)
Trajectory tracking	Model predictive control on tracking error model	Better results than classic state-tracking controller	yes	(Klančar & Škrjanc, 2007)
Trajectory tracking	Adaptive sliding mode control	Presence of parameter variations and disturbances	yes	(C.-Y. Chen et al., 2009)
Trajectory tracking	Linear interpolation method	Simple calculation to attain control signals	yes	(Scaglia et al., 2010)
Trajectory tracking	Backstepping and neural fuzzy network in polar coordinates	Presence of motor friction nonlinearity	yes	(Jang, 2011)
Trajectory tracking	Approximation-based adaptive neural control in polar coordinates	Presence of unknown skidding and slipping	no	(Yoo, 2012)
Trajectory tracking	Backstepping and Lyapunov redesign	Presence of unknown parameter variations and skidding	no	(Hwang et al., 2013)

2.2.4 Related works

A large amount of theoretical studies were conducted on motion control of mobile robots and car-like vehicles, among which only a few extended to experimental validations. The majority of the existing research dealt with small wheeled mobile robots with differential steering. Some practical investigations focused on motion control of agricultural vehicles and robots using nonlinear control methods, and they are introduced as follows.

Thuilot et al. (2002) designed a feedback control law for the steering angle of a farm tractor to realize automatic path following. The control law was derived based on the chained form of the nonlinear kinematic model. Fang et al. (2006) extended that work and designed a feedback control law for both the forward velocity and the steering angle to achieve trajectory tracking. The control law was derived using integrator backstepping, and a modified kinematic model which integrated the sliding effects as additive unknown parameters was constructed. Experimental tests were performed to validate the proposed control strategies.

Gomez-Gil et al. (2011) designed two new path following control laws for the steering angle of a farm tractor, one for following straight lines and one for following circular curves. The proposed control laws were derived based on the tractor kinematic model and proved by Lyapunov theory. The new singularity-free controller was superior to the one in Thuilot et al. (2002), and thus had better tracking properties when the tractor was placed with large initial orientation errors.

Tu (2013) designed a path following controller for a customized four-wheel drive, four-wheel steering agricultural robot based on its dynamic model. The controller was designed using backstepping-based sliding mode control. Simulation results verified the robustness of the proposed approach, which was insensitive to external disturbances and parameter variations. The same control approach was applied on the kinematic model for the trajectory tracking task. Experiments of tracking different shapes of trajectories were conducted to test the controller performance.

Bayar et al. (2015) designed a trajectory tracking controller for an orchard vehicle. The rear-wheel side slip errors and other unmodelled dynamics were represented by two additive terms as harmonic trigonometric functions. The effectiveness of the controller was tested in field experiments.

Lenain et al. (2017) designed an extended kinematic model with a side slip angle observer for accurate off-road path following of a mobile robot. Methods of adaptive control

and predictive control were applied in designing control laws for the steering of the robot. Using the proposed control strategies, the robot was able to follow off-road reference paths with arbitrary shapes.

2.2.5 Summary

Because of their wide range of applications, mobile robots and vehicles received intensive attention among researchers worldwide during the past two decades. The nonholonomic constraints restrict the motion capabilities of these platforms and this stimulated researchers' interests to design novel control strategies to solve various motion tasks. Studies at the beginning mainly focused on searching for stable control laws for the trajectory tracking, path following, and point stabilization problems, when the systems were under perfect conditions and kinematic models were looked for. Strategies including approximate and exact feedback linearization and smooth and nonsmooth time-varying feedback were applied. For trajectory tracking and path following tasks, the method of chained form representation combined with backstepping control and Lyapunov functions was well developed. The polar coordinates representation was applied in the point stabilization task. These methods were taken as foundations for many subsequent studies. Later on, research activities expanded the targets from perfect kinematic models to imperfect realities that existed in real robotic systems. Effects including wheel slipping and skidding, internal nonlinearities, external disturbances, input saturations and disturbances, and parameter variations were taken into account. Dynamic models were considered in some research about small-scale mobile robots. The imperfect effects were assumed as additive parameters in extended kinematic models or dynamic models, and robust controllers were designed to compensate those effects. Nonlinear robust control methods that were commonly used included sliding mode control, model predictive control, neural networks, and adaptive control. These methods provided the system with robustness to deal with external disturbances, system delays, large initial errors, which frequently existed in practical applications.

Although research on motion control of mobile robots and vehicles is abundant, the motion control of the WSIC remains a new task thanks to its unique operation patterns. The existing literature provides a series of classic control models and algorithms, which can be referred to for the development of control models and algorithms for the WSIC.

Chapter 3

Control Models and Algorithms

3.1 Introduction

Automatic guidance of the WSIC platform is a process in which the two WSIC support vehicles automatically navigate on the defined paths and cooperate with each other under properly designed control algorithms to complete the desired field operations. Control models and algorithms are key elements in an automatic guidance system. In this chapter, two different control algorithms were designed for the WSIC, one for its mobile operating mode and one for its stationary operating mode. For WSIC operations, synchronization of the two vehicles is a crucial requirement. To achieve synchronized motions of the two vehicles, a master-slave cooperative method was applied for both operating modes.

Guidance algorithms are designed based on mathematical models of both the vehicles and the guidance tasks. To derive an appropriate control model for the WSIC, a compromise must be reached between a complex model which takes all the geometrical, kinematic, and dynamic relationships into account and a very simple model which includes many assumptions and neglects some of these relationships. A complex model can precisely describe the machine's behaviour, but its complexity may limit the design of effective control laws. On the other hand, a simple model is easy to manage from a control design point of view, but its effectiveness may not be maintained under real field conditions.

When designing high-performance guidance systems for off-road farm vehicles, a thorough understanding of vehicle dynamics, in both longitudinal and lateral directions, is essential. Longitudinal dynamics involves the analysis of friction and traction forces on the tires and the analysis of load distributions and transfers on the front and rear axles along the vehicle's direction of travel. Lateral stability is a concern when an off-road vehicle has to make large steering manoeuvres or travel on slippery and unpredictable terrains.

The lateral forces acting on the wheels when turning and lateral disturbances can cause side slips of the vehicle motions. In most cases, the farm fields for vehicles to travel on have flat terrains. Even when vehicles have to traverse on fields with undulations, the amplitude of vehicle's vertical motion is normally negligible compared with its horizontal displacement. Thus, it is reasonable to assume that off-road farm vehicles move on 2D surfaces (Rovira Más et al., 2010, Chapter 2). Longitudinal dynamics plays an important role when the vehicle travels on a steep slope. It is crucial to calculate the forces on the front and rear tires to make sure that the vehicle has reasonable tractions and load distributions on all tires. The influence of longitudinal dynamics is more on operation efficiency rather than operation accuracy since the longitudinal slip does not cause lateral deviations. When designing guidance systems for farm vehicles to work on flat and firm grounds, the dynamic behaviours of the vehicles are usually ignored and kinematic models are used. In previous research on automatic guidance of farm vehicles, the application of kinematic models is well established. The developed guidance systems and control strategies were tested to be effective as long as desired conditions including flat travel paths and reasonable operation velocities were met and appropriate guidance appliances were used (Gomez-Gil et al., 2011; Nagasaka et al., 2004; Thuilot et al., 2002). However, the lateral dynamics has significant effects on the guidance accuracy when the vehicle is required to travel on slopes or slippery terrains, or to follow irregular paths under high operating speeds. Under such conditions, the high performance achieved in those previous studies using pure kinematic designs cannot be maintained and the compensation of sliding effects is necessary. This compensation requires the development of an extended kinematic model or a dynamic model. In some studies, different non-ideal operation conditions were considered and the lateral slips were modelled, such as Cariou et al. (2009), in which a kinematic model extended with sliding parameters was developed for a mobile robot to travel on a slope, Fang et al. (2011), in which a kinematic model extended with sliding parameters was developed for a mobile robot to travel under lateral disturbances and high speeds, and Lucet et al. (2015), in which an extended kinematic model combined with a dynamic side slip model was developed for a mobile robot to travel on a slippery surface under high speeds.

The two tractors of the WSIC can be treated as independent power units and can be controlled separately, as long as their relative distance is within the sliding range of the connecting truss. The WSIC has a long span and a heavy structure, which makes the full analysis of its dynamic behaviours much more complex than that of an independent farm vehicle's. However, WSIC operations are usually performed under the following conditions: (1) regular-geometry fields and straight, firm travel paths; (2) no major differences in elevation between both ends of the machine; (3) low operating speeds. Under these conditions, the WSIC can be treated as a machine travelling on a 2D surface with a large

payload. Since the WSIC is not operated on a slope under high speeds, the weight of the truss can be treated as vertical loads on the two tractors, and its influence on the tractors' lateral stability is negligible.

In this research work, kinematic models of the WSIC are looked for with some reasonable assumptions being made: (1) both vehicles have independent control mechanisms and the control inputs are linear velocities and front-wheel steering angles; (2) the WSIC is assumed to move on flat and firm paths, so roll and pitch of the vehicles can be ignored; (3) the WSIC moves according to the pure rolling and non-slipping assumption, which is generally the case in practice because most WSIC operations are slow-speed. Based on the above assumptions, the kinematic control models for both operating modes were developed and the control laws for the velocity and steering angle inputs were designed.

3.2 Mobile mode

When the WSIC works under the mobile mode, the implement is placed in parallel with the span of the truss at a given location, and the entire machine is displaced along the length of the field. When the machine finishes one pass, the implement is displaced by one working width and then the process is repeated. The mobile mode is ideal for fertilizing operation when the spraying appliances span the entire truss and the work of one field is finished by one single pass. Figure 3.1 illustrates the operating process of the mobile mode. The two WSIC tractors continuously travel along two parallel paths. To make sure that the operation is accurate, the two vehicles need to move forward at the same pace, i.e., one vehicle should not significantly lead or lag behind the other.

3.2.1 System modelling

The control tasks for the mobile mode include guiding the two tractors on their individual paths and synchronizing their forward motions. Since the two WSIC tractors can be controlled separately, a path following model can be developed for both tractors, and an additional synchronous tracking model can be applied to one of the tractors to achieve synchronization control. Overall, the mobile mode is defined as a synchronous trajectory tracking problem.

Both WSIC tractors are typical four-wheeled vehicles with front-wheel steering. A commonly used model for this type of vehicle is the bicycle model, as shown in Figure 3.2 (De Luca et al., 1998). The equation of motion can be written as

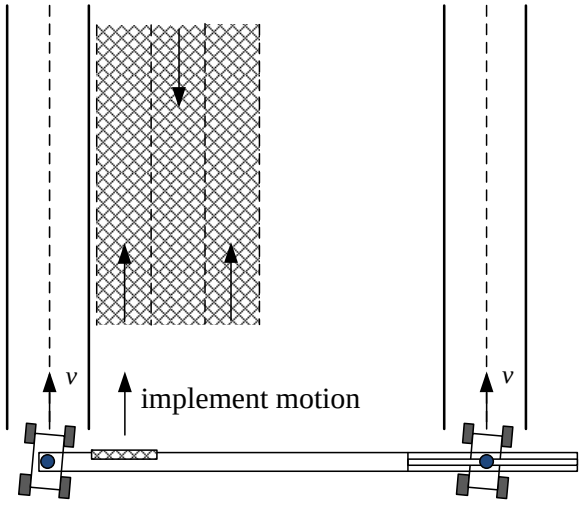


Figure 3.1: Mobile operating mode of the WSIC platform.

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = v\frac{\tan\delta}{l} \end{cases} \tag{3.1}$$

where v is the driving velocity, δ is the steering angle of the front wheel with respect to the forward direction of the vehicle, (x, y) is the position of the midpoint o of the rear wheel axis, θ is the orientation of the vehicle with respect to the X-axis, and l is the length of the wheelbase.

Under the mobile mode, the vehicles need to follow their predefined paths individually and keep their motions in synchronization in the meantime. They are controlled to move forward at an almost identical velocity. This velocity is determined according to the specific operation that must be performed by the WSIC. When the WSIC runs on a rectangular field, the two paths for the two vehicles can always be oriented, without loss of generality, to be parallel with the Y-axis, so that the offset between the two vehicles can be defined in the direction of the Y-axis. As shown in Figure 3.3, the offset error e is the difference of the two vehicles' Y coordinates. To eliminate the offset error, a master-slave cooperative control method can be applied. One of the vehicles (left one in Figure 3.3) is taken as the master, and its main task is to follow the left path with a predetermined operation velocity. The slave vehicle to the right has to track the real-time positions of the master to eliminate the offset by necessary velocity adjustments while at the same time following its reference path by proper steering control.

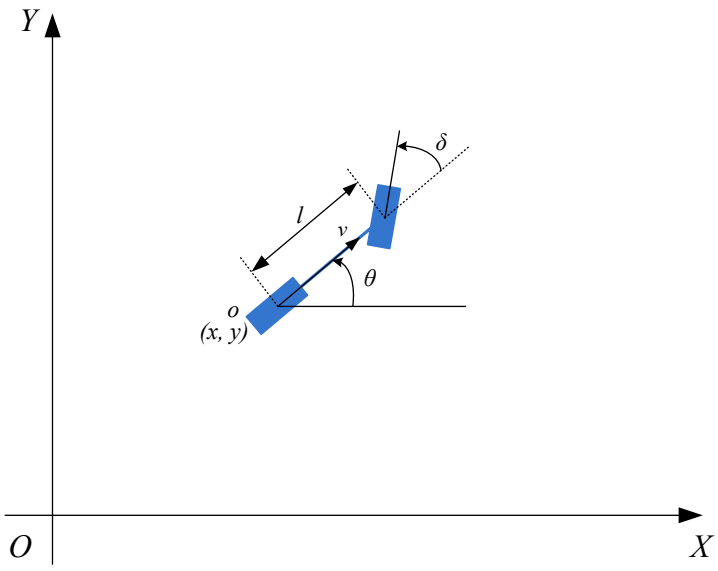


Figure 3.2: Bicycle model of a car-like vehicle in Cartesian coordinates.

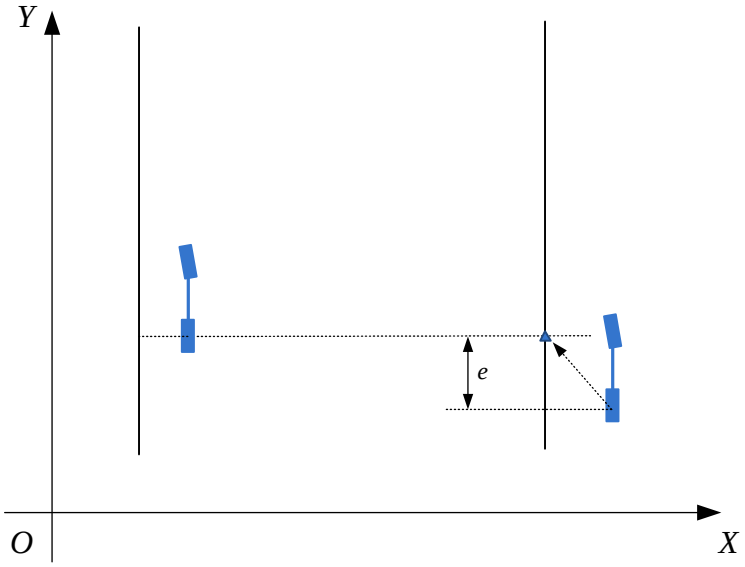


Figure 3.3: Offset of WSIC vehicles.

For both the master and the slave vehicles, the steering angle is controlled to eliminate the lateral deviation to the reference path. It belongs to a path following problem of the motion control tasks introduced in the literature review. The kinematic model for the path following problem is shown in Figure 3.4 with the relevant variables appearing in the model denoted as follows:

- C: the path for the vehicle to follow,

- M: the reference point on C which is the closest to the centre of the vehicle virtual rear wheel o,
- $(x,\ y,\ \theta)$: the coordinates of the vehicle o with respect to the inertial frame (OXY),
- s: the curvilinear coordinate of point M along C, and $c(s)$ denotes the curvature of C at that point,
- d: the lateral error of the vehicle to the reference point M,
- θ_t: the tangent of the path at point M with respect to the X-axis,
- θ_p: the orientation error of the vehicle with respect to the path C, $\theta_p = \theta - \theta_t$,
- δ: the steering angle of the virtual front wheel,
- v: the linear velocity of the vehicle with respect to the inertial frame,
- l: the vehicle wheelbase.

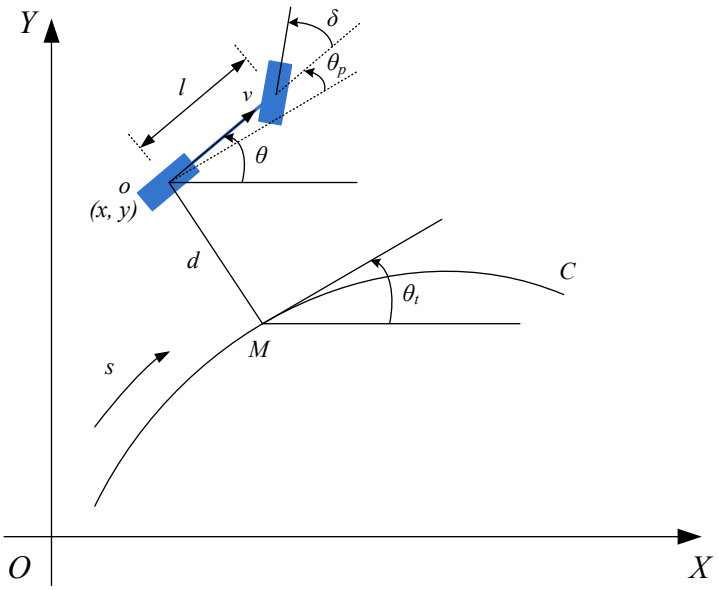


Figure 3.4: Kinematic model of a path following vehicle.

In this model, C is a generalized path represented by the curvilinear coordinate s and the curvature $c(s)$. The vehicle motion states can be represented using the path coordinates (s, d, θ_p) and the following equations can be derived (De Luca et al., 1998; Thuilot et al., 2002):

$$\dot{\theta}_t = \frac{\dot{s}}{1/c(s)} = \frac{v\cos\theta_p}{\frac{1}{c(s)} - d} \tag{3.2}$$

$$\dot{\theta}_p = \dot{\theta} - \dot{\theta}_t \tag{3.3}$$

$$\dot{d} = v\sin\theta_p \tag{3.4}$$

Combined with equation (3.1), the kinematic model equation in terms of path coordinates

is given as

$$\begin{cases} \dot{s} = \frac{v\cos\theta_p}{1-dc(s)} \\ \dot{d} = v\sin\theta_p \\ \dot{\theta}_p = v\left(\frac{\tan\delta}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right) \end{cases} \tag{3.5}$$

The model (3.5) is not defined when $d = 1/c(s)$, i.e., the vehicle is right at the instantaneous rotation centre of the path. For the case of WSIC operations, this is not encountered because the two reference paths of the field are parallel straight lines with near zero curvatures and the vehicles are always close to the paths. When the paths are straight lines, i.e., $c(s) = 0$, a simpler model is obtained as

$$\begin{cases} \dot{s} = v\cos\theta_p \\ \dot{d} = v\sin\theta_p \\ \dot{\theta}_p = v\frac{\tan\delta}{l} \end{cases} \tag{3.6}$$

For the slave vehicle, the forward velocity is controlled to track the instantaneous position of the master vehicle. A virtual reference vehicle can be assumed to be moving on the path and its position is the instantaneous projection of the master. As shown in Figure 3.3, the offset error e is the perpendicular distance between the virtual reference and the position of the slave vehicle. In the path coordinates, e is defined as the difference of coordinate s between the two vehicles, and the model equation can be derived as

$$\dot{e} = v_m\cos\theta_{p_m} - v_s\cos\theta_{p_s} \tag{3.7}$$

where e is defined as positive when the master leads the slave and subscripts m and s stand for master and slave respectively. The slave vehicle velocity v_s is controlled to eliminate the offset error e.

3.2.2 Control design for the master vehicle

For the master vehicle, the steering angle δ is controlled to eliminate the lateral error d. Since d is not directly related to δ in model (3.6), taking the second-order derivative of d yields

$$\ddot{d} = v^2\cos\theta_p \cdot \frac{\tan\delta}{l} \tag{3.8}$$

Thus a direct relationship between d and δ is generated. The acceleration term $\dot{v}\sin\theta_p$ is dropped on the right side of equation (3.8) because the forward velocity v is normally kept as a positive constant in WSIC operations. For equation (3.8), using the method of

feedback linearization, let

$$v^2 \cos\theta_p \cdot \frac{\tan\delta}{l} = -k_1 v\dot{d} - k_2 v^2 d \tag{3.9}$$

where $k_1, k_2 > 0$ are user-defined control gains, then we have

$$\ddot{d} + k_1 v\dot{d} + k_2 v^2 d = 0 \tag{3.10}$$

It is easy to verify from Routh's stability criterion that the steady-state d will be stabilized to zero as long as coefficients $k_1 v > 0$ and $k_2 v^2 > 0$. From equation (3.9), the steering control law can be designed as

$$\frac{\tan\delta}{l} = \frac{-k_1 v\dot{d} - k_2 v^2 d}{v^2 \cos\theta_p} = -k_1 \tan\theta_p - k_2 \frac{d}{\cos\theta_p} \tag{3.11}$$

or

$$\delta = \arctan\left(l\left(-k_1 \tan\theta_p - k_2 \frac{d}{\cos\theta_p}\right)\right) \tag{3.12}$$

When the reference paths are curved, the model equation (3.5) should be used. By applying the same design procedures, the following steering control law can be designed

$$\delta = \arctan\left(l\left(-k_1 \tan\theta_p - k_2 \frac{d}{\cos\theta_p} + \frac{c(s)\cos\theta_p}{1 - dc(s)}\right)\right) \tag{3.13}$$

which incorporates the curvature $c(s)$ of the path.

In some previous works about automatic guidance of agricultural vehicles, the following empirical steering control law was applied (Nagasaka et al., 2009, 2004; Stoll & Kutzbach, 2000; C. Zhang et al., 2016)

$$\delta = -k_\theta \theta_p - k_d d \tag{3.14}$$

which did not consider the vehicle model. In essence, equation (3.14) is a simplification of formula (3.11). When the vehicle is close to the reference path with small angular deviations, the following approximation can be assumed

$$\tan\theta_p \approx \theta_p, \cos\theta_p \approx 1 \tag{3.15}$$

and $\frac{\tan\delta}{l}$ can be approximated by a linear term $k\delta$ when the steering angle is in the range [-45°, 45°]. From equation (3.10), the critical damping condition for the control model requires

$$(k_1 v)^2 - 4k_2 v^2 = 0 \tag{3.16}$$

which is equivalent to

$$k_1 = 2\sqrt{k_2} \tag{3.17}$$

Equation (3.17) provides a guideline for choosing control gain values.

3.2.3 Control design for the slave vehicle

Since the steering control law equation (3.12) is velocity independent, the same law can be applied to the slave vehicle for its path following task and the slave vehicle velocity can be controlled for the motion synchronization purpose. For the synchronous tracking problem modelled by equation (3.7), let

$$v_m \cos\theta_{pm} - v_s \cos\theta_{ps} = -k_e e \tag{3.18}$$

where $k_e > 0$ is a third user-defined control gain, then we have

$$\dot{e} + k_e e = 0 \tag{3.19}$$

and e will go to zero asymptotically if equation (3.18) holds in the operating process. The following velocity control law is suggested by equation (3.18)

$$v_s = \frac{v_m \cos\theta_{pm} + k_e e}{\cos\theta_{ps}} \tag{3.20}$$

The control structure of the mobile mode is shown in Figure 3.5. The master vehicle performs a path following task under a reference velocity v_r. The real-time motion states of the master vehicle including its position, orientation, and velocity are sent to the slave vehicle, and a virtual reference (x_r, y_r, θ_r) is generated on the reference path. The slave vehicle controls both its steering angle and its velocity to perform a path following and a synchronous tracking tasks.

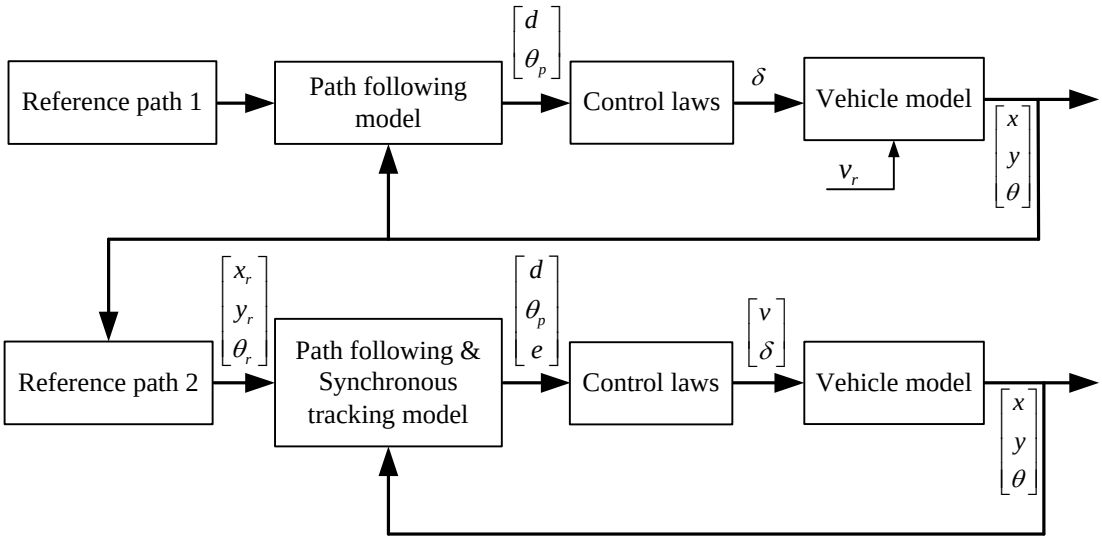


Figure 3.5: Control structure of the mobile mode.

3.3 Stationary mode

Stationary mode is the main operating mode of the WSIC and it has been used in pruning and harvesting operations in cranberry production. Under the stationary mode, the implement is placed vertically to the span of the truss. The field operations are completed along the span of the truss while both tractors are at rest. The machine travels forward by a distance corresponding to the working width of the implement after the mobile carriage finishes one pass across the field. Figure 3.6 illustrates the operating process of the stationary mode. To achieve satisfactory operation accuracy, both tractors need to move forward by an equal distance and stop precisely at desired locations on the paths. In addition, to make sure that the operation is efficient, the forward moving processes of the tractors should be performed in an approximately synchronized manner.

3.3.1 System modelling

The control tasks for the stationary mode involve a periodic point tracking task for both vehicles and a coordination task between the two vehicles. The stationary mode can be defined as a parallel point-to-point tracking problem. The vehicles are static while the implement is operating along the truss. When the implement finishes one pass across the field, the vehicles start moving forward. After both vehicles reach desired locations and stop, the implement starts a new pass across the field. The flexibility of the WSIC structure allows the two vehicles to be controlled independently and stop at the desired

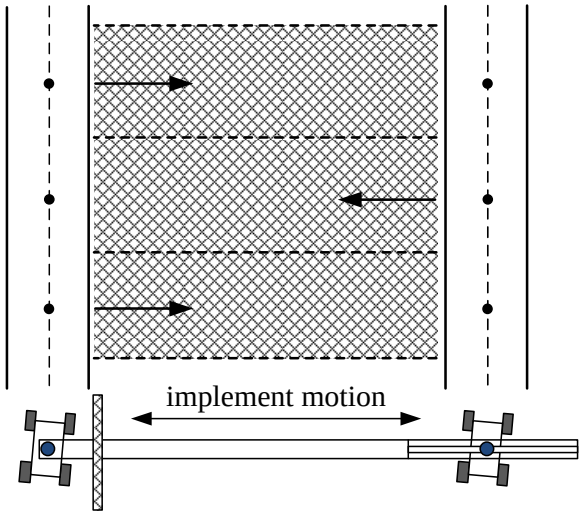


Figure 3.6: Stationary operating mode of the WSIC platform.

locations with a short time difference. However, the implement can only start operating when both vehicles reach a full stop. In actual operations, skillful operators can keep constant communication and control the two vehicles in a synchronized manner to keep high operation efficiency. The automation of the stationary mode requires the following two problems to be solved: (1) a point tracking control problem for both WSIC vehicles; (2) a sequential control problem for the WSIC to complete the operation cycle.

For the point tracking problem, the vehicles need to move forward and stop at desired locations on the paths in each operation cycle. This can be modelled as a point tracking control problem for both vehicles. Given that the WSIC tractors are independent power units and the forward process is slow-speed, the problem is modelled at a kinematic level and the linear velocity and the steering angle are deemed as control inputs. Figure 3.7 depicts a point tracking problem for a car-like vehicle with the relevant variables denoted as follows:

- $(x,\, y)$: the coordinates of the vehicle o with respect to the inertial frame (OXY),
- θ: the orientation of the vehicle with respect to the X-axis,
- $(x_d,\, y_d)$: the coordinates of the desired position with respect to the inertial frame,
- θ_d: the orientation at the desired position with respect to the X-axis,
- δ: the steering angle of the virtual front wheel,
- v: the linear velocity of the vehicle with respect to the inertial frame,
- l: the vehicle wheelbase.

The control task is to move the vehicle from a start point (x, y, θ) to a desired target (x_d, y_d, θ_d).

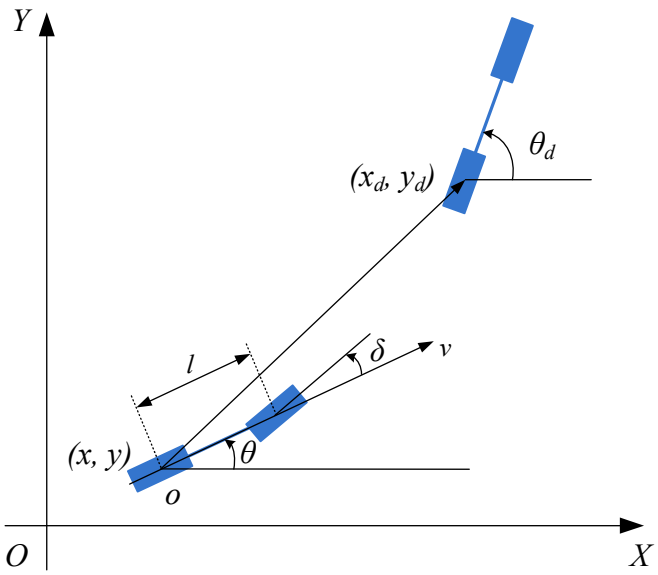


Figure 3.7: Kinematic point tracking model for WSIC vehicles.

For the sequential control problem, the WSIC is required to complete a series of steps and repeat the same steps in every operation cycle. When the implement finishes one pass, two new target points are assigned to the vehicles, and then both vehicles start the forward point tracking process. When both vehicles reach the target points and stop, the implement can start a new round of operation. To ensure that the operation is efficient, the vehicles should start and stop moving at approximately the same time.

3.3.2 Design of forward point tracking

To guide the WSIC vehicles from a start point to a forward target, proper velocity and steering angle inputs need to be designed. Since the WSIC is a large and heavy platform, the forward point tracking process needs to be smooth and continuous under mild accelerations. The point tracking problem, also referred as point stabilization or set point regulation, can be solved by three well-known control laws: time-varying smooth feedback, non-smooth feedback, and time-invariant smooth feedback based on polar coordinates transformation. It has been shown that design with polar coordinates can achieve fast and smooth performance for the forward parking problem (Oriolo et al., 2002), which is analogous to the point tracking problem for the stationary mode of the WSIC. Figure 3.8 depicts the point tracking problem in polar coordinates with the additional variables denoted as follows:

- ρ: the linear distance between the vehicle start point and the desired target,
- ψ: the angle of the distance line with respect to the X-axis,

- α: the angle of the distance line with respect to the vehicle velocity line,
- β: the angle of the distance line with respect to the desired orientation.

After the following coordinates transformations

$$\rho = \sqrt{(x_d - x)^2 + (y_d - y)^2} = \sqrt{\Delta_x^2 + \Delta_y^2} \tag{3.21}$$

$$\psi = ATAN2\left(\Delta_y, \Delta_x\right) \tag{3.22}$$

$$\alpha = \psi - \theta \tag{3.23}$$

$$\beta = \psi - \theta_d \tag{3.24}$$

in which $ATAN2(\Delta_y, \Delta_x) \in$ (-180°, 180°] is the four-quadrant inverse tangent function, the kinematic model equation in terms of polar coordinates can be written as (Aicardi et al., 1995; Indiveri, 1999)

$$\begin{cases} \dot{\rho} = -v\cos\alpha \\ \dot{\alpha} = -v\left(\frac{\tan\delta}{l} - \frac{\sin\alpha}{\rho}\right) \\ \dot{\beta} = v\frac{\sin\alpha}{\rho} \end{cases} \tag{3.25}$$

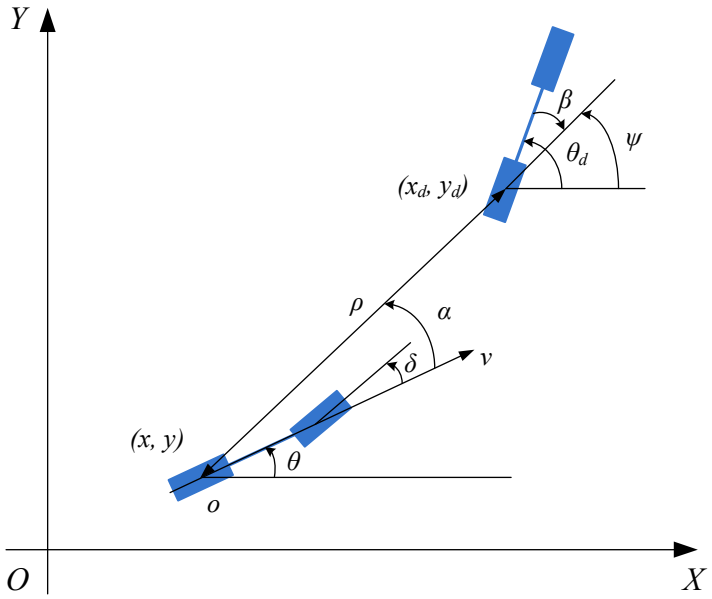


Figure 3.8: Kinematic point tracking model in polar coordinates.

In model (3.25), ρ, α, and β are defined as the distance error, heading error, and orientation error, respectively. The point tracking task of reducing the differences $(x_d - x, y_d - y, \theta_d - \theta)$ to zero in the Cartesian coordinates is transformed into eliminating the

errors (ρ, α, β) in the polar coordinates. For the case of a WSIC operation, the two vehicles need to track target points which are defined on two straight paths. The point tracking tasks for the WSIC vehicles are forward tracking, and the two vehicles have to keep their tracks in the allowable path width with minimum lateral deviations. When the WSIC vehicle is placed at the start point of the path, it is reasonable to assume that the heading and orientation errors are well within the range (-30°, 30°).

To realize smooth forward point tracking for the WSIC vehicles, two control strategies are designed, one for the forward velocity and the other for the steering angle. For efficiency consideration, the vehicles are expected to approach the target points with forward motions only, i.e., no backward movements or negative velocities are allowed.

(1) Velocity control

When a target point is determined on the path, the distance error can be used as a dependent variable for velocity control. A classic velocity control law for the point stabilization problem is in the form of $v = k\rho$ (Astolfi, 1999; Indiveri, 1999) or $v = k\rho\cos\alpha$ (Aicardi et al., 1995; Oriolo et al., 2002) where k is a positive control gain. Under such velocity control, the vehicle adjusts its forward speed proportionally to the distance error. When the vehicle gets closer to the target, its output speed gets smaller. However, the vehicle has to generate the maximum speed at the start of the motion and then monotonically decrease the speed to zero when it reaches the target. This sudden and great acceleration motion at the start is not practical and should be avoided for large agricultural platforms.

In this research work, a new form of velocity control law is designed. The basic control strategy is that the vehicle accelerates to maximum speed when it reaches the midpoint of the distance and slows down to zero speed when it reaches the target. The velocity control law is designed as

$$v = \left(\frac{-2^n(\rho - 0.5\rho_0)^n}{\rho_0^n} + 1 \right) k_v, (n = 2, 4, 6...) \tag{3.26}$$

where ρ_0 is the initial distance error determined by the start point and the target point, k_v is a positive velocity coefficient which determines the maximum velocity during the tracking process, and n is a positive even order number which reflects the change rate of the speed. Higher values of n result in higher acceleration and deceleration cycles. Equation (3.26) is drawn in Figure 3.9. The velocity profiles are in the general form of parabola curves. The exact form of the velocity equation (the values of k_v and n) should be determined according to the initial distance error and the maximum acceleration that the WSIC platform permits.

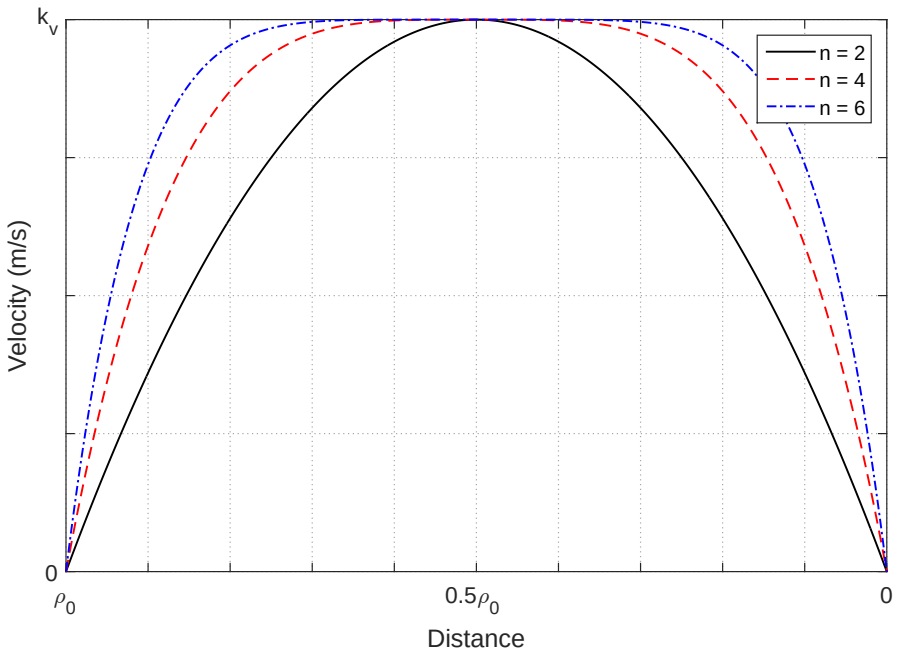


Figure 3.9: Velocity profiles with respect to the distance error.

(2) Steering angle control

The steering angle δ is controlled during the tracking process to correct the heading error α and the orientation error β. The Lyapunov-like technique was applied in some early studies to design steering strategies for the general point stabilization problem (Astolfi, 1999; Indiveri, 1999). One typical form of the steering control laws is

$$\frac{\tan\delta}{l} = c = \frac{\sin\alpha}{\rho} + k_1\frac{\beta}{\rho}\frac{\sin\alpha}{\alpha} + k_2\frac{\alpha}{\rho} \tag{3.27}$$

where k_1 and k_2 are positive control gains. Such steering control laws are designed for the vehicles to track any target points specified on a 2D plane and the vehicles are allowed to travel freely to approach the target through any possible trajectory. For WSIC operations, the target points are on a straight path that has a limited width, the initial heading and orientation errors when the WSIC is placed on the field paths should be kept as small as possible and the vehicles have to limit their trajectories within the width of the reference paths.

In this research work, a new steering control law based on the backstepping method is designed. The control strategy is to eliminate the orientation error β before the distance error ρ reaches zero so that the vehicle always approaches the target point in straight line

tracking. From the model equation (3.25), only the heading error is directly controlled by the steering angle. To correct the orientation error, the backstepping method is applied and detailed as follows.

For the kinematic model (3.25), consider a Lyapunov function candidate for the first step as

$$V_1 = \frac{1}{2}\sin^2\beta \tag{3.28}$$

Its derivative along equation (3.25) is

$$\dot{V}_1 = \sin\beta\cos\beta\cdot\dot{\beta} = \sin\beta\cos\beta\cdot\frac{v\sin\alpha}{\rho} \tag{3.29}$$

To make $\dot{V}_1$ negative definite, regard $u_1 = \sin\alpha$ as the virtual control input and choose it as

$$u_{1d} = -k_1\sin 2\beta \tag{3.30}$$

where $k_1 > 0$ is a user-defined control gain, then we have

$$\dot{V}_1 = \sin\beta\cos\beta\cdot\frac{v}{\rho}\left(-k_1\sin 2\beta\right) = -\frac{k_1 v}{2\rho}(\sin 2\beta)^2 \leq 0 \tag{3.31}$$

As v and ρ are always positive before the vehicle reaches the target point, this permits that β is decreased to zero before the velocity and distance error go to zero. If u_1 tracks u_{1d} precisely, then β will converge to zero asymptotically. Indeed, in the closed-loop system, u_1 is not the actual control input and tracks u_{1d} with some error. This error is defined as

$$\tilde{u}_1 = u_1 - u_{1d} \tag{3.32}$$

Computing the derivative of $\tilde{u}_1$ yields

$$\dot{\tilde{u}}_1 = \cos\alpha\cdot\dot{\alpha} + 2k_1\cos 2\beta\cdot\dot{\beta} \tag{3.33}$$

For step two, a Lyapunov function candidate is chosen as

$$V_2 = V_1 + \frac{1}{2}\tilde{u}_1^2 \tag{3.34}$$

Its derivative along equations (3.29) and (3.33) is

$$\dot{V}_2 = -\frac{k_1 v}{2\rho}(\sin 2\beta)^2 + \tilde{u}_1\left(\frac{v\sin 2\beta}{2\rho} + \cos\alpha\left(-\frac{v\tan\delta}{l} + \frac{v\sin\alpha}{\rho}\right) + 2k_1\cos 2\beta\frac{v\sin\alpha}{\rho}\right) \tag{3.35}$$

In equation (3.35), if $\frac{\tan\delta}{l}$ is chosen as

$$\frac{\tan\delta}{l} = c = \frac{\sin\beta\cos\beta}{\rho\cos\alpha} + \frac{\sin\alpha}{\rho} + \frac{2k_1\cos 2\beta\tan\alpha}{\rho} + k_u\tan\alpha + \frac{k_1 k_u \sin 2\beta}{\cos\alpha} \tag{3.36}$$

where $k_u > 0$ is a second user-defined control gain, then we have

$$\dot{V}_2 = -\frac{k_1 v}{2\rho}(\sin 2\beta)^2 - k_u v\tilde{u}_1^2 \leq 0 \tag{3.37}$$

From equation (3.36), we can obtain the following control law for the steering angle

$$\delta = \arctan(cl) \tag{3.38}$$

Using control law (3.36) or (3.38) in the closed-loop system, when β approaches zero, the motion equation for α becomes

$$\dot{\alpha} = -v\left(\frac{2k_1}{\rho} + k_u\right)\tan\alpha \tag{3.39}$$

It is easy to verify that α also tends towards zero asymptotically, since for a small angle α ($|\alpha| < 30°$), a Lyapunov function $V = \alpha^2/2$ with $\dot{V} = -v\,(2k_1/\rho + k_u)\,\alpha\tan\alpha \leq 0$ can be found.

(3) Modification of control strategies

According to equation (3.36), the steering angle input, in theory, will go to zero when the heading and orientation errors are decreased to zero, and this is expected to occur before the vehicle reaches the target point. However, to implement the steering control law on a practical vehicle, the limitations of sensing hardware need to be taken into account. The measurements of ρ, α, and β are sensor dependent and absolute zero errors cannot be achieved. When the vehicle approaches the target, the distance error ρ decreases to near zero, and measurements of α and β become unstable since α and β are defined based on the existence of ρ. The non-zero α and β, and the small ρ will result in erratic steering angle input near the end of the point tracking process. To avoid this behaviour, the steering control strategy is modified by extending the true target point to a virtual one. As shown in Figure 3.10, a virtual target point is located on the extension of the tracking trajectory. The steering input equation (3.36) is calculated from ρ', α', and β' instead. The purpose of this modification is to make angles α and β always well-defined when the vehicle approaches the true target. The principle of this modification is that the elimination process of the orientation error β is equivalent to that of β'. When α' and β' are decreased to zero, α

and β approach zero as well. Under this modification, the distance error from the start point to the true target is approximated by the perpendicular distance between the two parallel lines that pass through the two target points. The velocity control law (3.26) is to guarantee that the vehicle moves forward by an exact perpendicular distance equal to ρ. With proper choosing of control gains k_1 and k_u, it can be permitted that the vehicle approaches and stops at the true target in a straight line tracking manner or stops on the perpendicular line beside the target with a small lateral error.

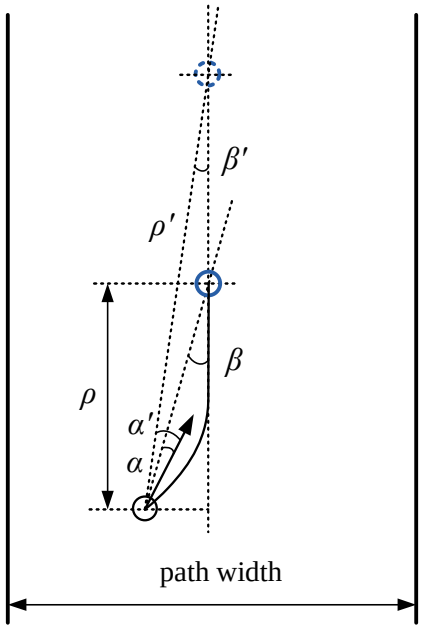


Figure 3.10: Modification of control strategies.

(4) Performance evaluation

The parameters ρ, α, and β used in the above control design are calculated from polar coordinates transformations. When the vehicle finishes a point tracking process and reaches the target point, the angles α and β become undefined, so these indirect parameters are not suitable for analyzing the tracking performance. To evaluate the tracking process, three errors are defined in the Cartesian coordinate system, as shown in Figure 3.11. When the vehicle stops at a point near the target, the discrepancy between the actual stop point and the desired target can be described by three error values: offset error e, lateral error d, and orientation error θ_e. The offset error e is the perpendicular distance from the actual stop point to the target. The lateral error d is the perpendicular distance from the actual stop point to the reference path. The orientation error θ_e is the angle difference between the

vehicle orientation and the desired target orientation. The errors drawn in the figure are defined as positive. For the stationary mode, the offset error e is the most important one since it directly relates to misses or overlaps for the field operation. The overall control goal of the stationary mode is to eliminate the offset error while minimizing the lateral and orientation errors.

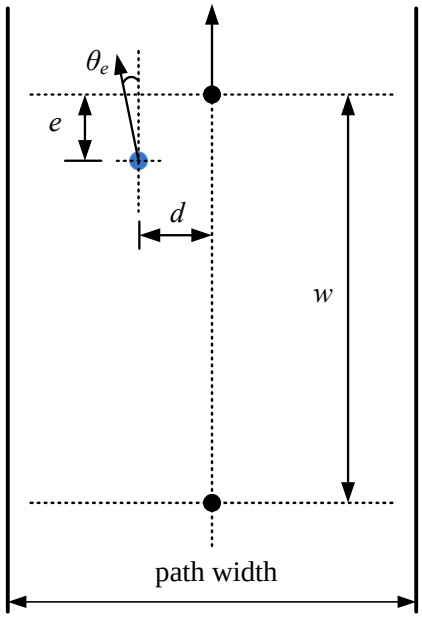


Figure 3.11: Point tracking errors for performance evaluation.

3.3.3 Design of control sequence

The operation of the stationary mode includes the following steps: determine the target points on the paths, start point tracking for both vehicles, stop both vehicles, start implement operation, and stop implement operation after one pass across the field is finished. In every operation cycle, a parallel point-to-point tracking process is performed. A master-slave cooperative control method could allow the WSIC to be operated by only one human operator. One vehicle with the operator onboard works as the master and controls all the operation sequences. The task of the slave is to follow the commands of the master to perform forward point tracking tasks.

The overall operation of the stationary mode can be divided into two stages: the preparation stage and the execution stage. In the preparation stage, the coordinates of the parallel field paths are obtained. The two paths are divided into small sections according to the width of the implement. Then a series of target points are generated and their

coordinates are calculated. An index number is added to each pair of target points as the control variable in the execution stage. During the execution stage, the control system runs a control sequence to accomplish the point tracking and implement operation tasks.

The designed control sequence is shown in Figure 3.12. The system starts with the index number initialized to zero. In the first step, the index number is updated and a pair of target points is determined. Then the master vehicle starts a point tracking task; meanwhile, it sends the same index number with a start signal to the slave. The slave receives the signal and then starts moving. During the point tracking process, the slave

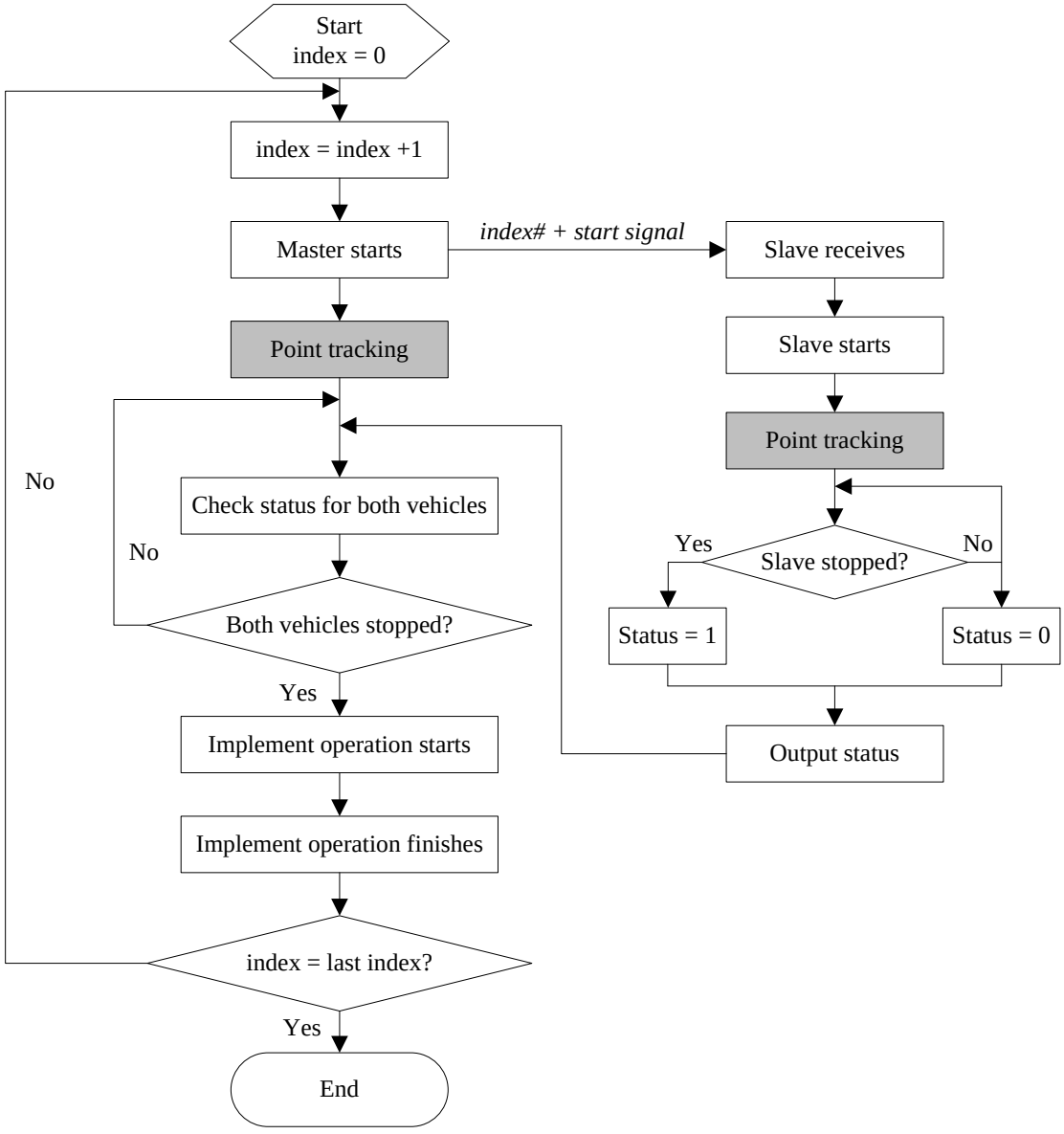


Figure 3.12: Control sequence of the stationary mode.

keeps sending its status to the the master. When the slave finishes the tracking process, it sends a stop signal (status = 1) back to the master. The control system keeps checking the status of both vehicles during the tracking process. When both vehicles are confirmed stopped, the implement is commanded to start operating. When the implement finishes one pass, the current index number is checked. If the last index is reached, the control sequence is ended, or the index is updated and the process is repeated.

3.4 Chapter summary

In this chapter, the control models were developed and the control algorithms were designed for the mobile and stationary operating modes of the WSIC. The mobile mode was modelled as a synchronous trajectory tracking problem, and a master-slave cooperative method was applied. For the master vehicle, a path following task was defined and a steering angle control law was designed. For the slave vehicle, a path following task and a synchronous tracking task were defined, and control laws were designed for both the steering angle and the forward velocity. During the operation, the motion states of the master including its position, orientation, and velocity are sent to the slave for synchronization control.

The stationary mode was modelled as a parallel point-to-point tracking problem, and a second master-slave cooperative method was applied. Two control tasks, one for the forward point tracking and the other for the periodic operating process, were defined. Control laws of the steering angle and the forward velocity were designed for the point tracking task and a control sequence was designed for the periodic operating process. During the operation, the master vehicle controls the start of the forward point tracking processes and the start of implement operations. The slave vehicle performs point tracking processes under the control signals sent from the master.

Chapter 4

Simulations

This chapter presents the simulation results for the designed control algorithms. For each operating mode, a MATLAB/Simulink model was developed to simulate the guidance models and control algorithms. The simulation results served to demonstrate the theoretical effectiveness of the control designs.

4.1 Mobile mode

4.1.1 Simulation model

Based on the control structure of the mobile mode (Figure 3.5), a MATLAB/Simulink model was built, as shown in Figure 4.1. The reference paths were defined in two user functions `Reference model_1` and `Reference model_2`, and path following and tracking errors were output from the two. The control laws were calculated in S-functions `master_ctrl` and `slave_ctrl`. The kinematic models of the master and slave vehicles with initial conditions were defined in another two S-functions `master_plant` and `slave_plant`. The positions of the vehicles, the errors, and the control inputs were saved into the MATLAB workspace for quick plotting and analysis during the simulation.

Three simulations were performed for the mobile mode. In the first two simulations, the two reference paths were defined as straight lines as $x = 0$ and $x = 30$, parallel to the Y-axis of the Cartesian coordinate system (x and y coordinates are in metres, the same below). The initial positions of the master and slave vehicles were assumed to be located at $(-0.5, 0, 90°)$ and $(30.6, -0.5, 105°)$ respectively. Thus, the initial lateral and orientation errors were 0.5 m, 0° for the master and -0.6 m, 15° for the slave, and the initial offset error between the two vehicles was 0.5 m.

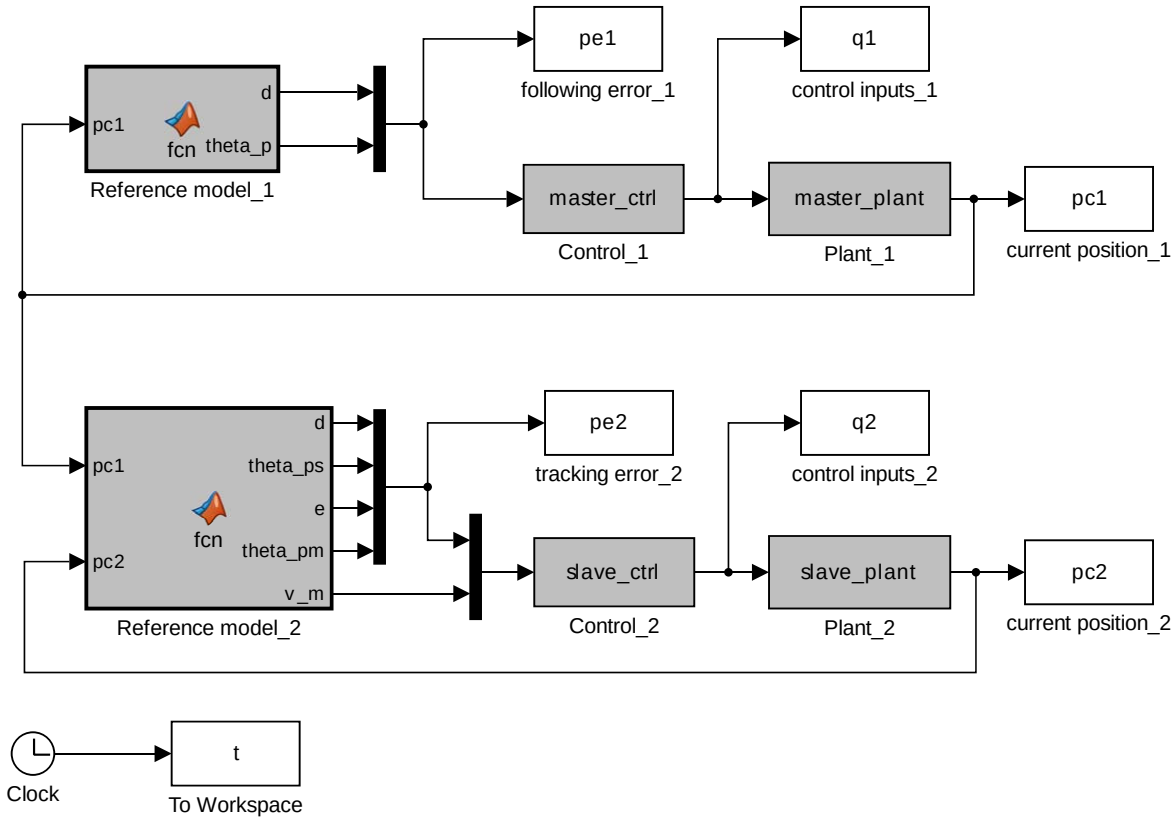


Figure 4.1: Simulink model for the mobile mode.

In the third simulation, the reference paths were defined as parallel curves. This simulation aimed to test the effectiveness of the control laws in following non-ideal paths under error disturbances. In all the simulations, the wheelbase of the vehicles was set to 3.2 m and the steering range was set to $\pm 35°$.

4.1.2 Constant reference velocity

In the first simulation, the reference velocity was set to a constant of 1.5 m/s. The control gains for the steering law were set to $k_1 = 1.2$ and $k_2 = 0.36$. These values were determined so that the resulted vehicle trajectory did not overshoot and the steering angle input was within the allowed limit. The control gain $k_e = 0.4$ was used for the slave velocity control.

The resulted vehicle trajectories are shown in Figure 4.2, and the variations of the lateral and orientation errors are shown in Figures 4.3 and 4.4 respectively. The steering angle inputs are presented in Figure 4.5. As these figures show, both vehicles could effectively correct the initial lateral and orientation deviations under the designed steering control.

Both the lateral and orientation errors and the steering inputs went to zero within 10 s. The offset error between the two vehicles is shown in Figure 4.6 and the velocity inputs are shown in Figure 4.7. The offset error was diminished to zero in less than 15 s. Since the slave was behind the master when the simulation started, its velocity input was slightly greater than the master's at the beginning. After the offset was eliminated, the two vehicles reached synchronized motions and their velocities were kept the same.

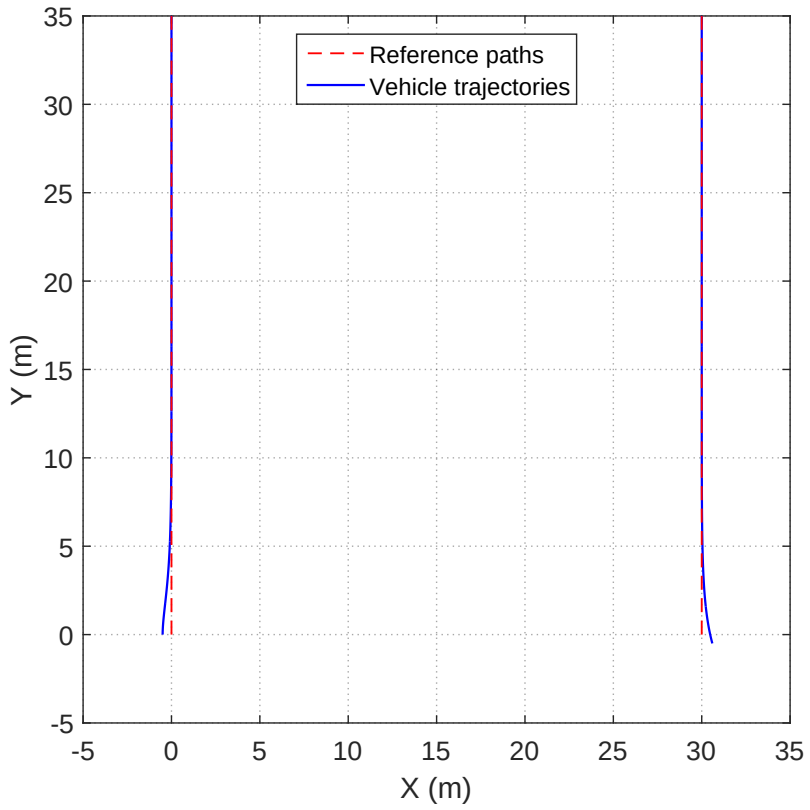


Figure 4.2: Synchronous tracking trajectories, constant reference velocity.

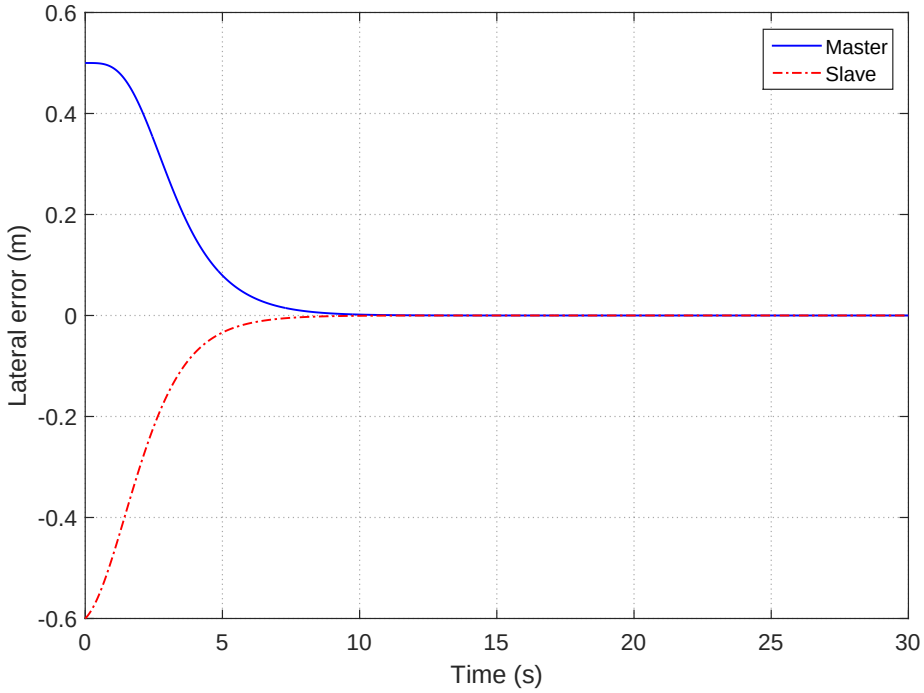


Figure 4.3: Lateral errors, constant reference velocity.

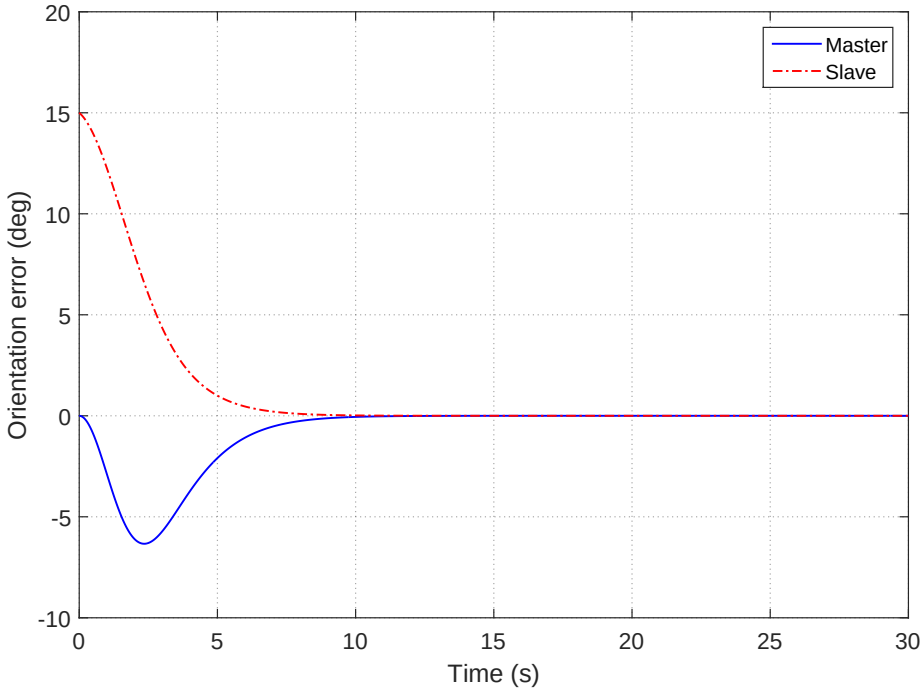


Figure 4.4: Orientation errors, constant reference velocity.

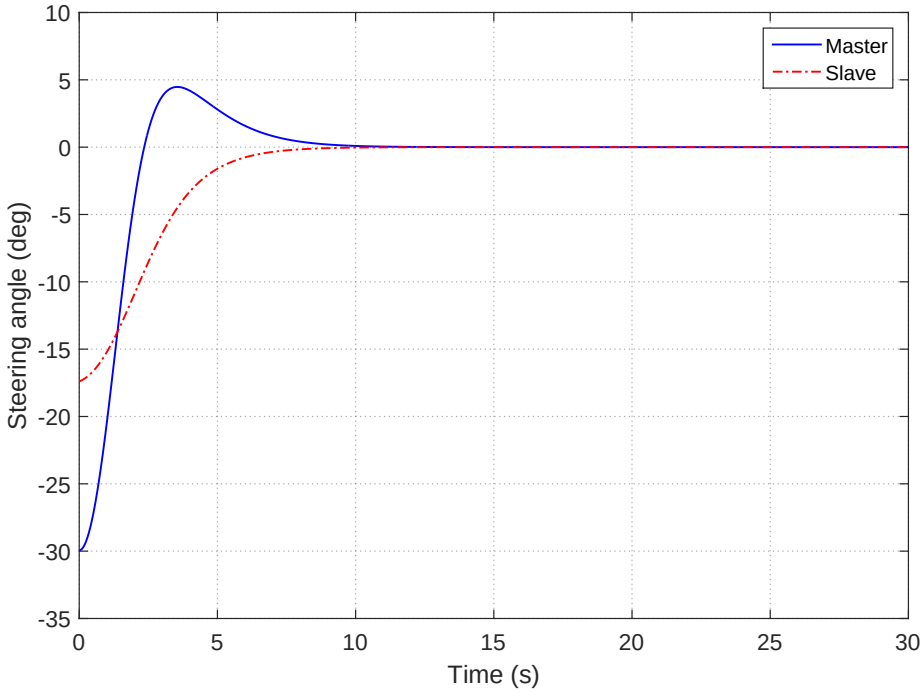


Figure 4.5: Steering angle inputs, constant reference velocity.

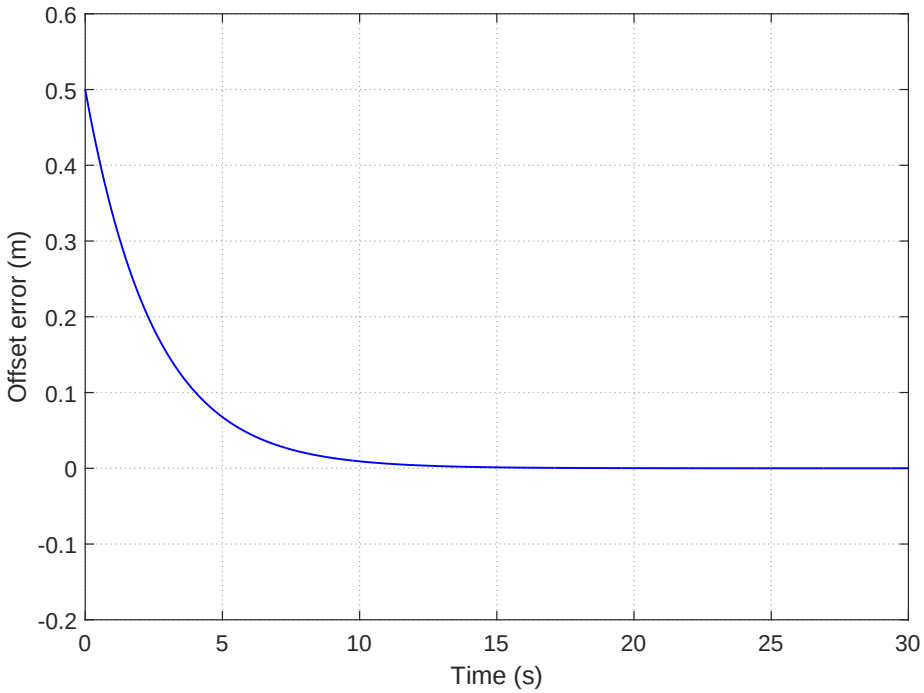


Figure 4.6: Offset error, constant reference velocity.

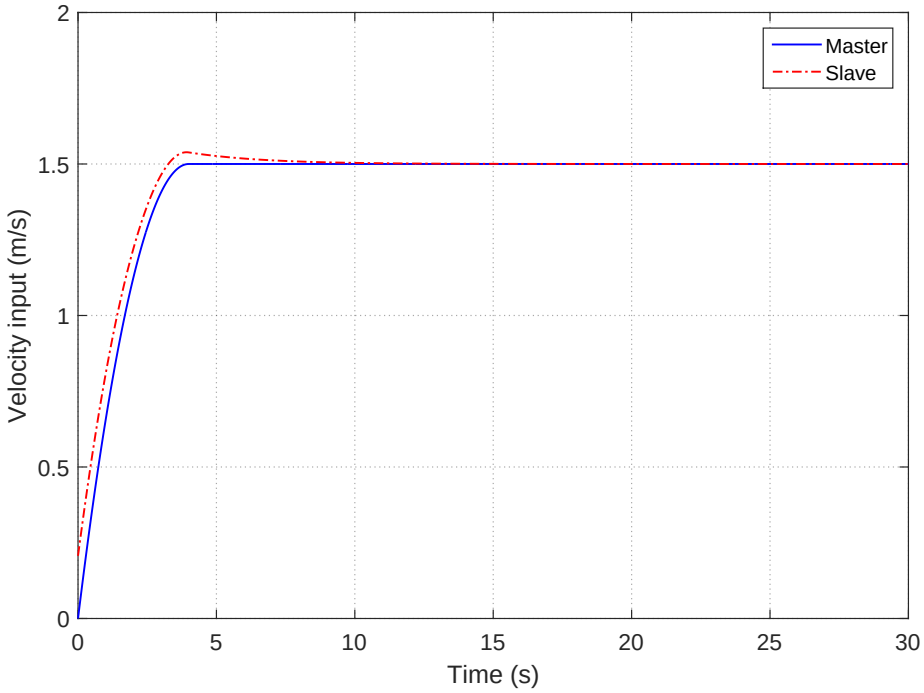


Figure 4.7: Velocity inputs, constant reference velocity.

4.1.3 Varying reference velocity

In the second simulation, a varying reference velocity was applied. The velocity input for the master vehicle was a sinusoidal curve with values ranging from 1.2 to 1.6 m/s. The same control gains as the ones used in the first simulation were used. The vehicle trajectories are plotted in Figure 4.8. The lateral errors, orientation errors, and the steering inputs are shown in Figures 4.9, 4.10, and 4.11, respectively. These results are very similar to the ones obtained in the constant reference velocity simulation. However, since the initial acceleration stage for the master vehicle in this simulation was slower than that in the constant velocity simulation, it took more time in this simulation (about 12 s) than in the constant velocity simulation (about 10 s) for the vehicles to correct the lateral and orientation errors. The offset error is shown in Figure 4.12 and the velocity inputs of the two vehicles are compared in Figure 4.13. As the figures show, the velocity input of the slave kept tracking the variations of the master velocity, and this permitted that motion synchronization between the two vehicles was always achieved.

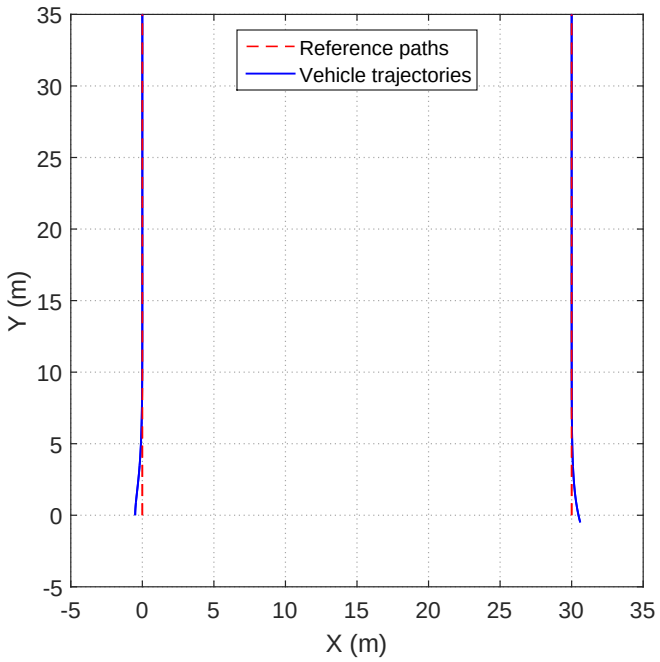


Figure 4.8: Synchronous tracking trajectories, varying reference velocity.

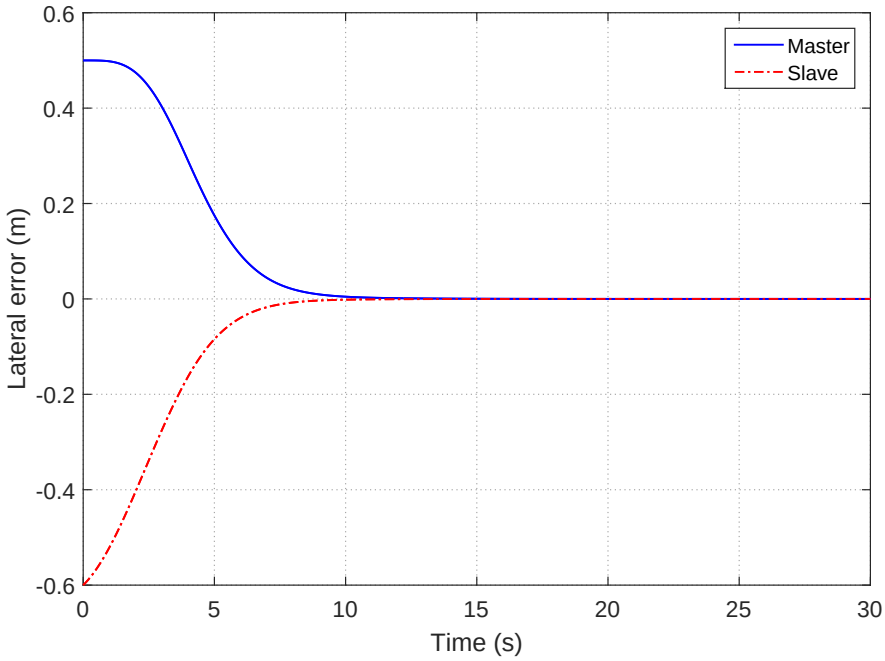


Figure 4.9: Lateral errors, varying reference velocity.

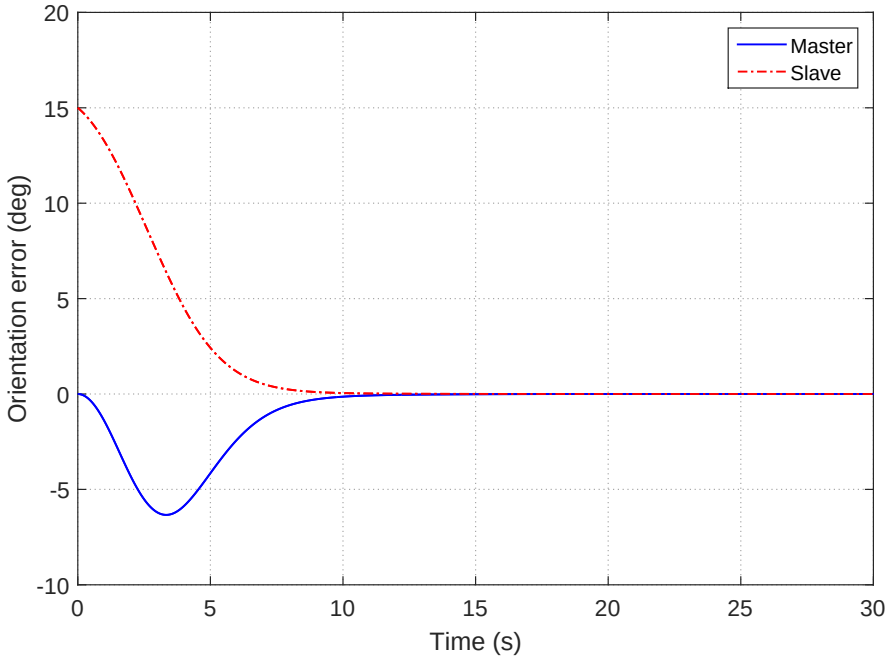


Figure 4.10: Orientation errors, varying reference velocity.

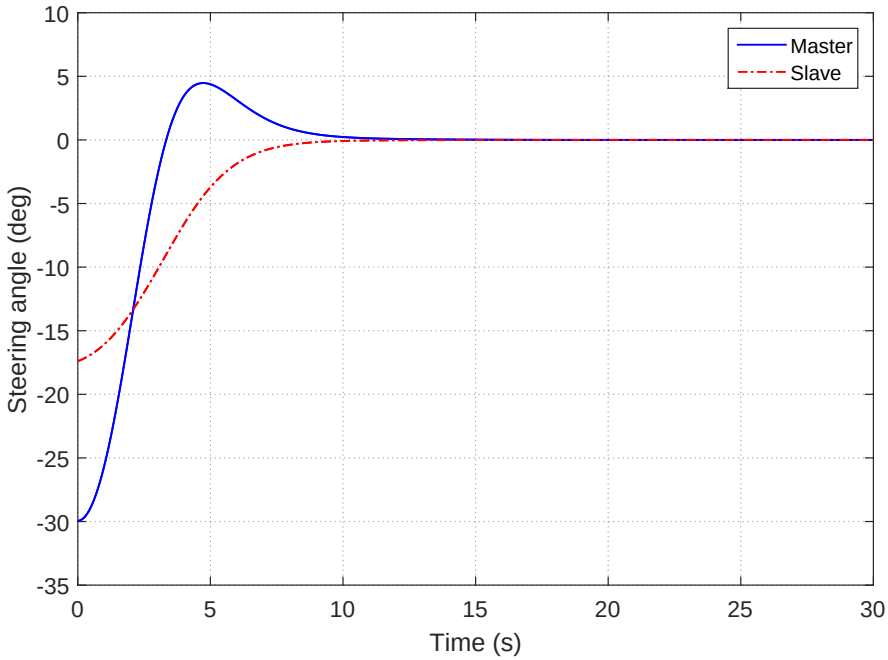


Figure 4.11: Steering angle inputs, varying reference velocity.

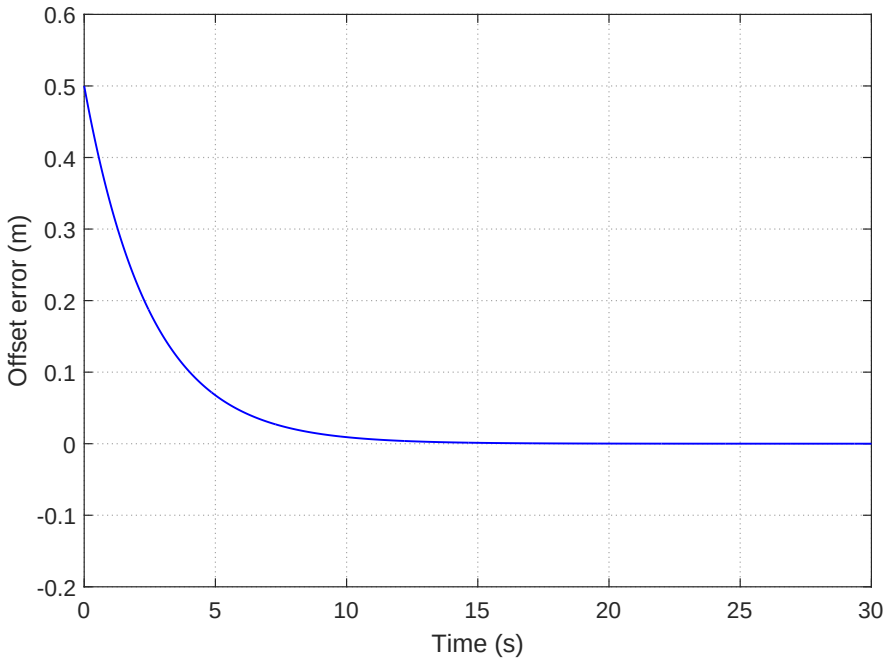


Figure 4.12: Offset error, varying reference velocity.

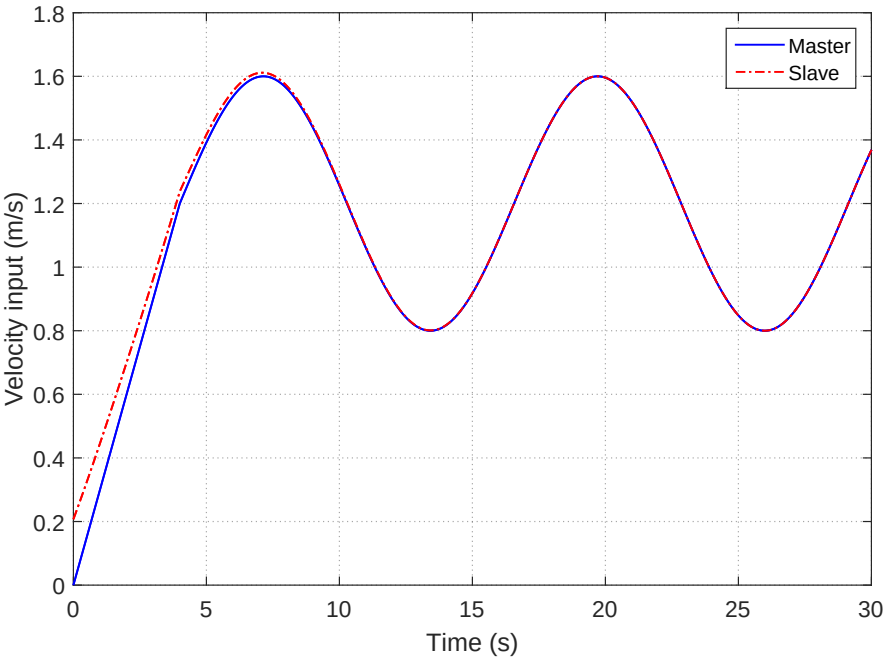


Figure 4.13: Velocity inputs, varying reference velocity.

4.1.4 Curved reference paths

In the third simulation, the reference paths were defined as two arcs having a radius of 50 m, the path for the master passing through points $(0, 0)$, $(10, 30)$ and the path for the slave passing through $(30, 0)$, $(40, 30)$. The reference velocity was set to 1.5 m/s and the control gains were set to $k_1 = 1.0$, $k_2 = 0.25$ and $k_1 = 1.2$, $k_2 = 0.36$ for the master and slave respectively. The control gain $k_e = 0.5$ was used for the slave velocity control. The initial positions of the master and slave vehicles were assumed to be located at $(-0.5, 0, 60°)$ and $(30.6, -0.5, 75°)$ respectively. Thus, the initial lateral and orientation errors were 0.4 m, 7.2° for the master and -0.78 m, 21.9° for the slave, and the initial offset error between the two vehicles was 0.5 m.

To compare the performance of the designed steering control law with that of the empirical linear steering control law (equation (3.14)), the path following task of the master vehicle using both steering control strategies was simulated. The control gains in equation (3.14) were set to $k_d = 0.7$ and $k_\theta = 2.4$. Figure 4.14 shows the lateral errors under the two control strategies. A constant offset of 0.09 m was presented under the linear steering control since it did not take into account the nonlinearity of the path. Another drawback of the linear steering control law was the intuitive tuning of the two control gains, which took a longer time than that for the proposed nonlinear control.

In the first two simulations, the initial errors of the vehicles were corrected and maintained at zero during the whole simulation process. To demonstrate the effectiveness of the control system at correcting unpredictable errors during the operation, a sudden position change was commanded for the master in the middle of the simulation.

The resulted vehicle trajectories are shown in Figure 4.15, and the variations of the lateral and orientation errors are shown in Figures 4.16 and 4.17 respectively. The steering angle inputs are presented in Figure 4.18. A sudden position change of the master occurred at $t = 30$ s in the simulation, which resulted a lateral error jump of 0.4 m. The steering control of the master responded appropriately to recover the vehicle from this sudden deviation. Figure 4.19 shows the offset error between the two vehicles. A small offset error was also generated when the master deviated from the reference path. However, as Figure 4.20 shows, the slave adjusted its velocity appropriately to correct this offset error back to zero.

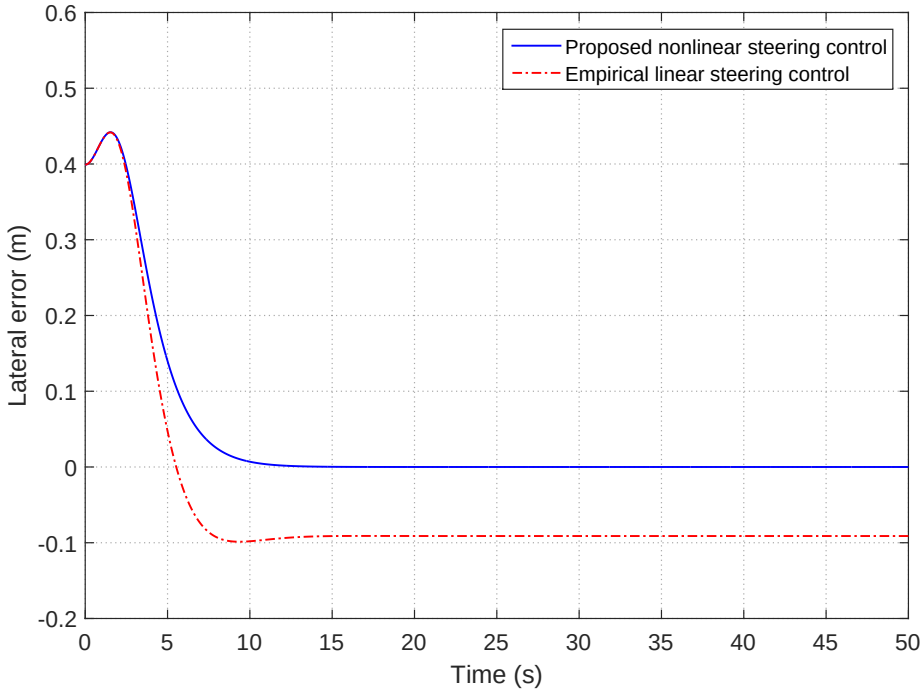


Figure 4.14: Lateral errors under proposed nonlinear steering control and empirical linear steering control.

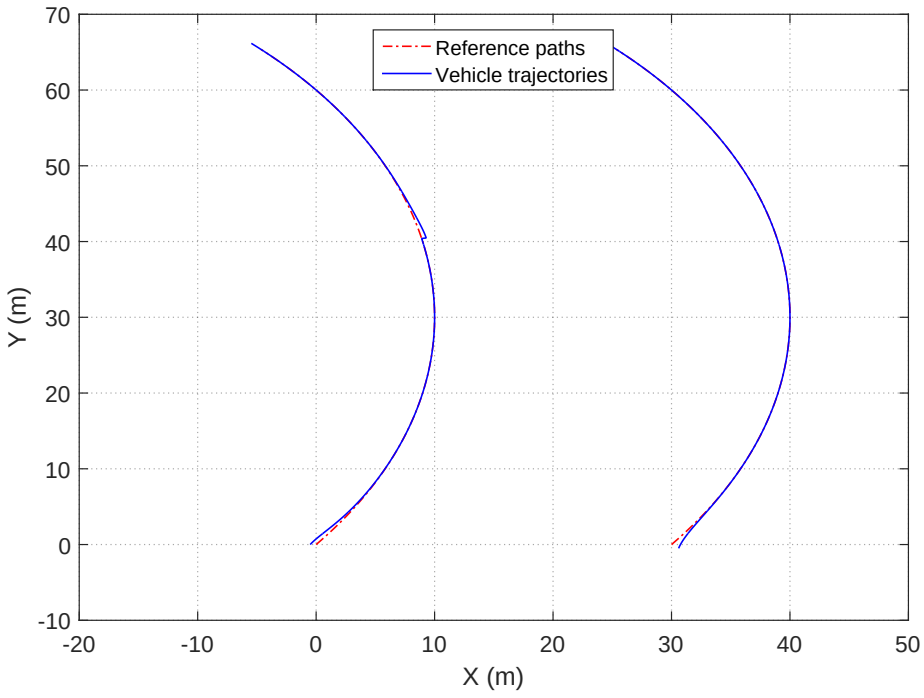


Figure 4.15: Synchronous tracking trajectories, curved paths.

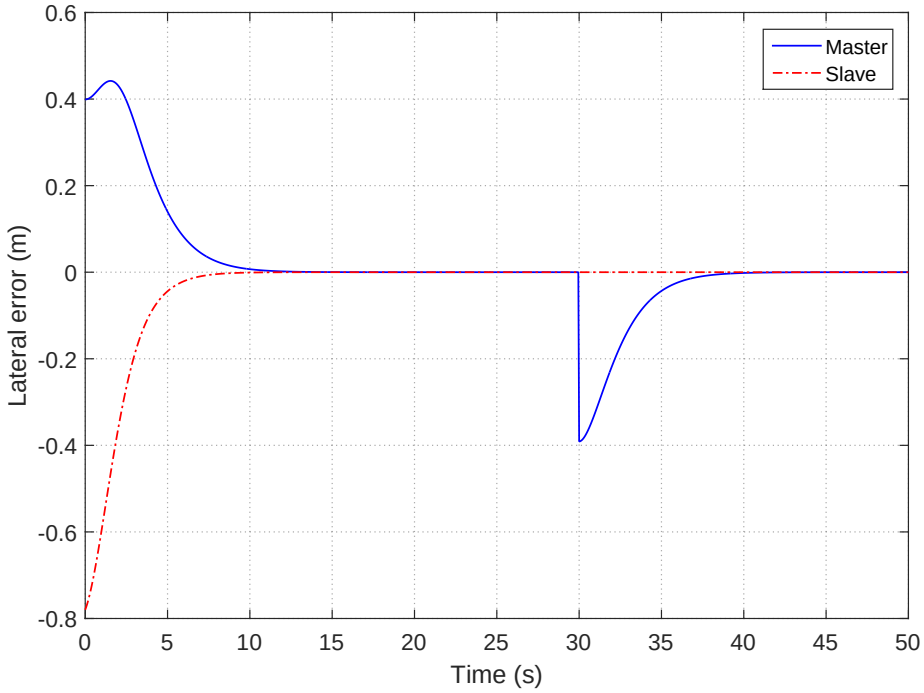


Figure 4.16: Lateral errors, curved paths.

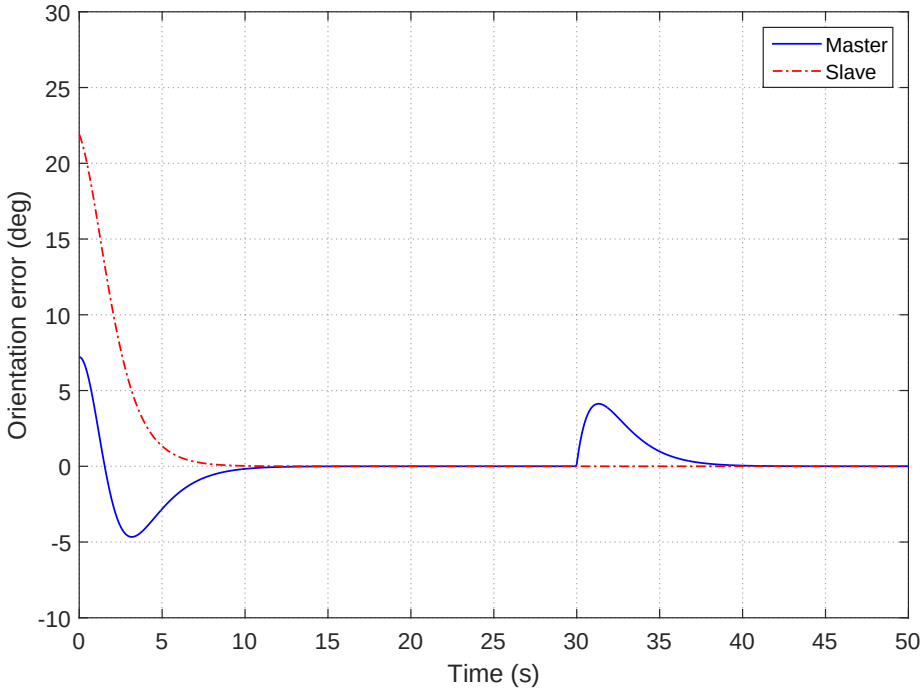


Figure 4.17: Orientation errors, curved paths.

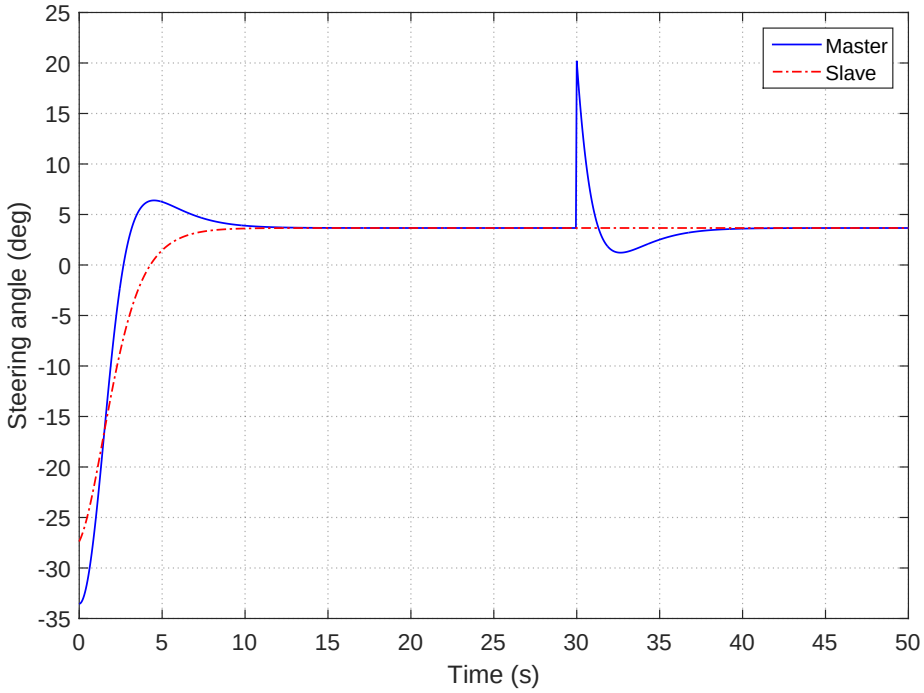


Figure 4.18: Steering angle inputs, curved paths.

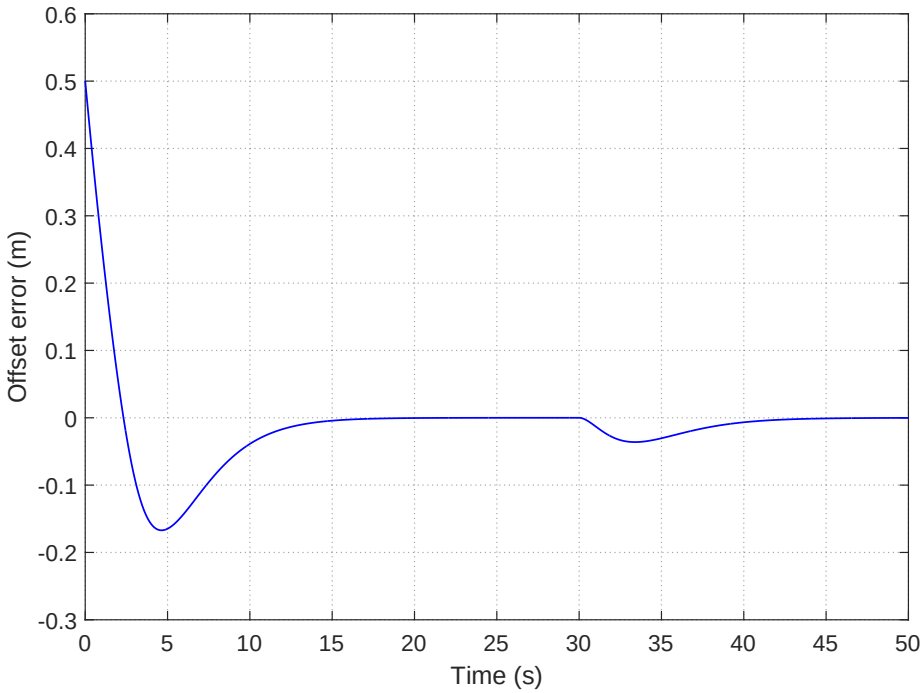


Figure 4.19: Offset error, curved paths.

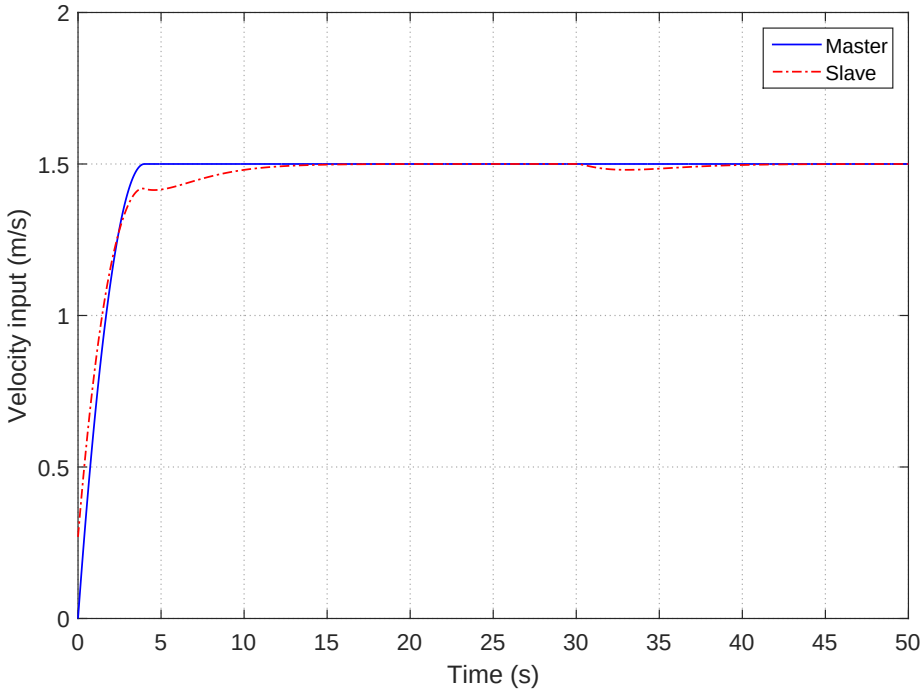


Figure 4.20: Velocity inputs, curved paths.

4.2 Stationary mode

4.2.1 Simulation model

Two groups of simulations were performed to validate the control algorithms for the stationary mode. First, the velocity and steering angle control laws were verified in a single point tracking case. Second, a sequential point-to-point tracking process was simulated by applying the designed control strategies in the control sequence. Figure 4.21 shows a MATLAB/Simulink model designed for the point tracking process. The tracking errors were calculated in a user function `Error calculation`, and the velocity and steering control laws were calculated in functions `Velocity law` and `Steering law`. The tracking process was stopped when the distance error `rho_perpen` of the modified control strategy reached the implement width.

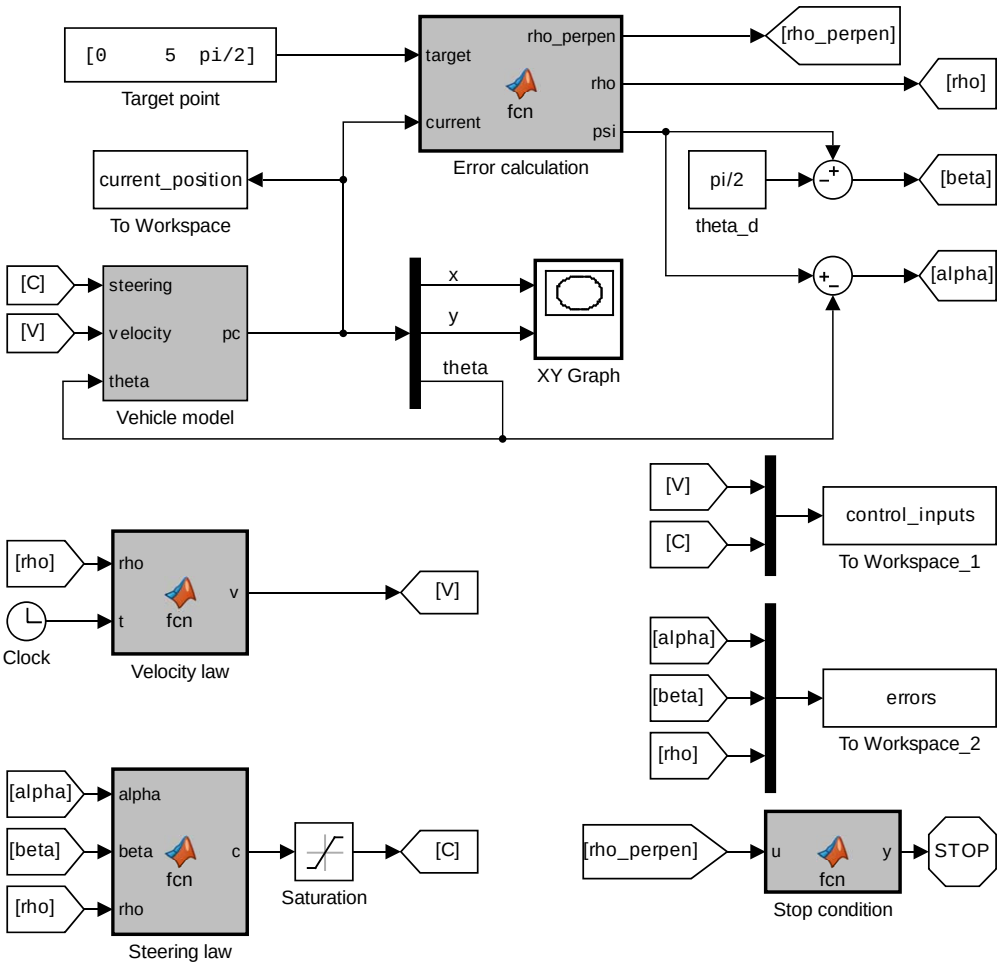


Figure 4.21: Simulink model for the stationary mode.

4.2.2 Single point tracking

To verify the proposed control strategies for the velocity and steering angle of the WSIC vehicles, the case of a single point tracking was simulated under different initial conditions. First, the exact form of the velocity control law (the values of k_v and n in equation (3.26)) was determined according to the implement width and the maximum acceleration allowed by the physical platform. Different implements can be mounted on the WSIC and their widths range from a few metres to more than 10 m. Two implement widths, 5 m and 10 m, were considered in the simulations. The suggested maximum acceleration was set to 1.1 m/s^2 for WSIC vehicles based on the acceleration tests for modern agricultural tractors (Meyer, 2010). The goal of this simulation was to determine a proper combination of k_v and n so that the tracking process was as efficient as possible while the resulted acceleration was below the limit. The initial heading and orientation errors were set to

zero in these simulations.

For a 5-m implement, two velocity coefficients were used, 1 m/s and 2 m/s. The resulted velocity and acceleration profiles are shown in Figure 4.22.

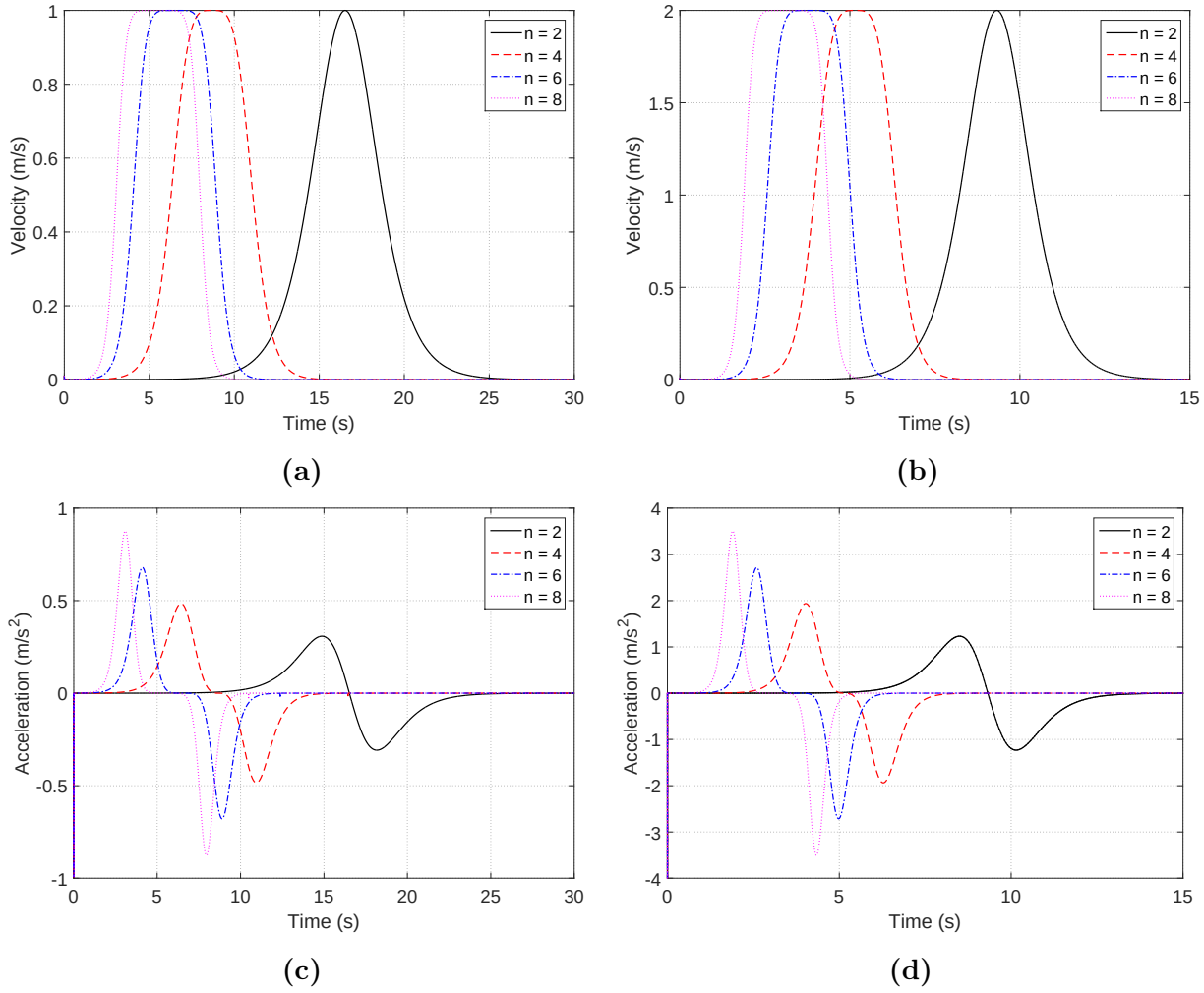


Figure 4.22: Velocity and acceleration profiles, (a) and (c) $\rho_0 = 5$ m, $k_v = 1$ m/s; (b) and (d) $\rho_0 = 5$ m, $k_v = 2$ m/s.

For a 10-m implement, two velocity coefficients were used, 2 m/s and 3 m/s. The resulted velocity and acceleration profiles are shown in Figure 4.23.

Comparing the time responses of the velocities and accelerations, when the implement width is 5 m, the values for k_v and n in the velocity control law should be determined as $k_v = 1$ m/s and $n = 8$, and when the implement width is 10 m, the values for k_v and n should be determined as $k_v = 2$ m/s and $n = 4$. For a different implement width, a similar simulation could be performed to determine appropriate values for k_v and n.

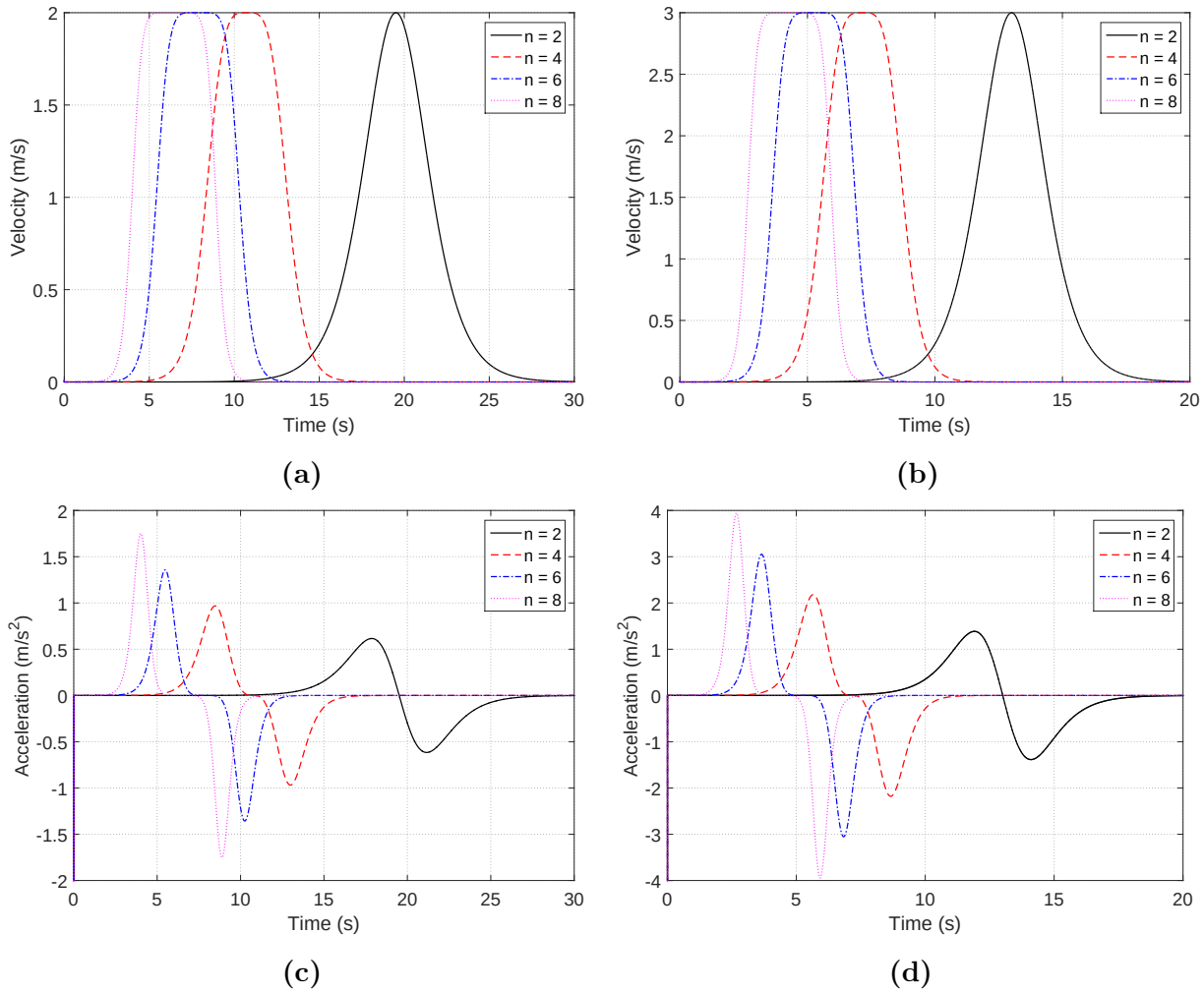
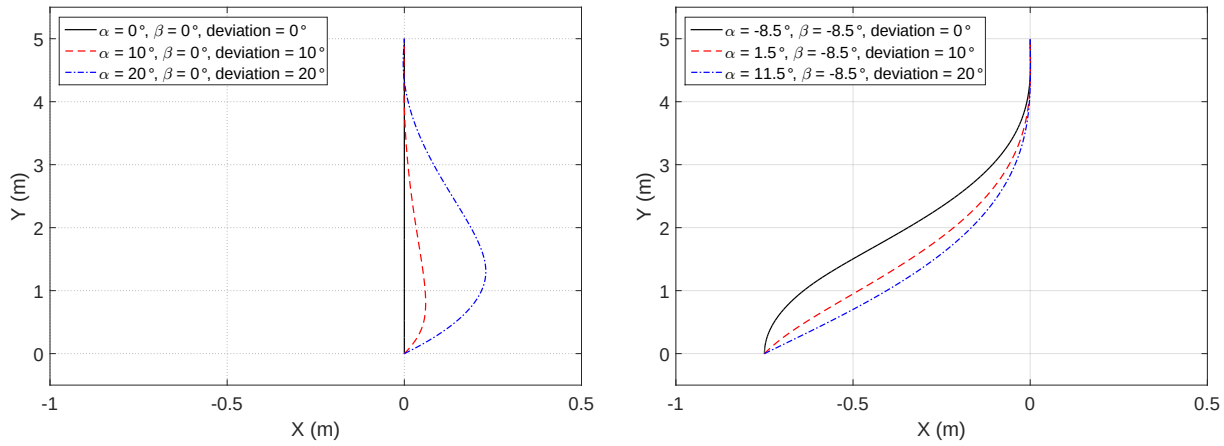


Figure 4.23: Velocity and acceleration profiles, (a) and (c) $\rho_0 = 10$ m, $k_v = 2$ m/s; (b) and (d) $\rho_0 = 10$ m, $k_v = 3$ m/s.

To test the effectiveness of the steering angle control law, the forward point tracking processes under different initial conditions were simulated and the trajectories are shown in Figure 4.24. The cranberry field paths have a minimum width of 3.4 m and the tractors used for the prototype WSIC have a tread width of 1.9 m (Laguë et al., 1997). This limits the maximum allowable lateral deviation of the tractor to the centreline of the path to be 0.75 m. For a 5-m implement, the maximum orientation error β is about 8.5°. When the WSIC is placed on the field paths, the heading deviations of the vehicles should be kept as small as possible. In the simulations, three heading deviation angles, 0°, 10°, and 20°, were considered. The steering range was set to ±35°, the control gains for the steering law (3.36) were set to $k_1 = 1.0$ and $k_u = 1.2$, and the velocity control law with $k_v = 1$ m/s and $n = 8$ was used. Under all situations, the vehicle adjusted the steering angle to correct the heading and orientation errors before the vehicle reached the target point.



(a) **(b)**

Figure 4.24: Point tracking trajectories, (a) zero lateral deviation; (b) 0.75-m lateral deviation.

For the case of a heading deviation of 20° and a lateral deviation of 0.75 m, the heading error α, orientation error β, and steering angle input δ with respect to the forward distance are shown in Figure 4.25. At the beginning stage, the vehicle steered to the left (positive steering angles) to correct the heading and orientation errors. When the lateral and angular deviations were decreased, the steering angle was decreased to small values around zero. This type of steering manoeuvre is natural and similar to steering by a human operator.

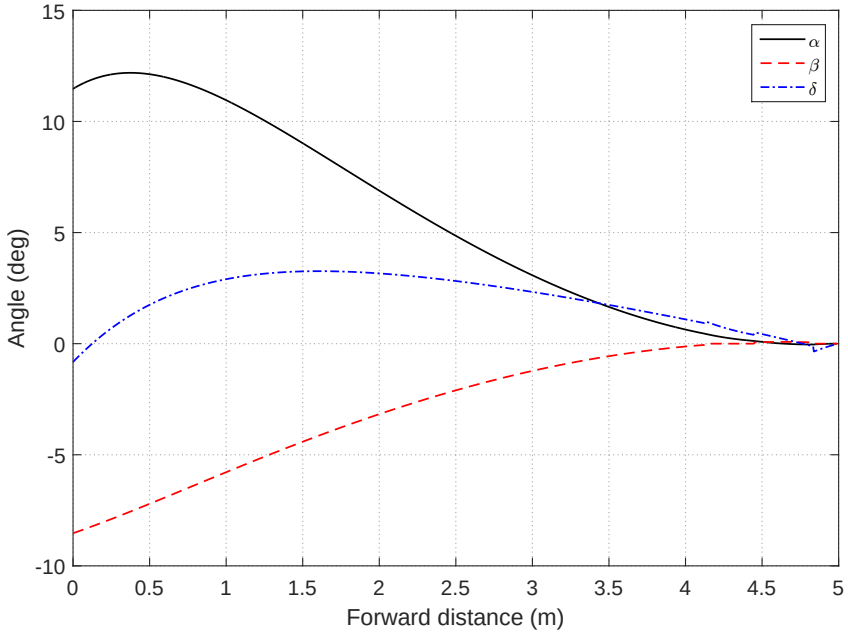


Figure 4.25: Heading error, orientation error, and steering angle input with respect to the forward distance.

In the above simulations, the ideal condition that angle errors α and β go to zero before ρ goes to zero was assumed. However, this is not to be achieved in real applications. To demonstrate the effectiveness of the modified control strategy, the case when the vehicle was placed with 0.75-m lateral deviation was simulated. The virtual target point was placed 1 m further to the true one. The control gains were kept unchanged in the simulation. Figure 4.26 shows the tracking trajectory, and Figure 4.27 shows the offset and lateral errors with respect to the forward distance. The variation of the orientation error θ_e is shown in Figure 4.28. The results show that when the vehicle stopped, the offset error was eliminated and the lateral error and orientation error were diminished to values near zero. The steering angle input is shown in Figure 4.29. The vehicle first steered to the maximum right to correct the lateral deviation and then gradually decreased the steering angle to a value close to zero.

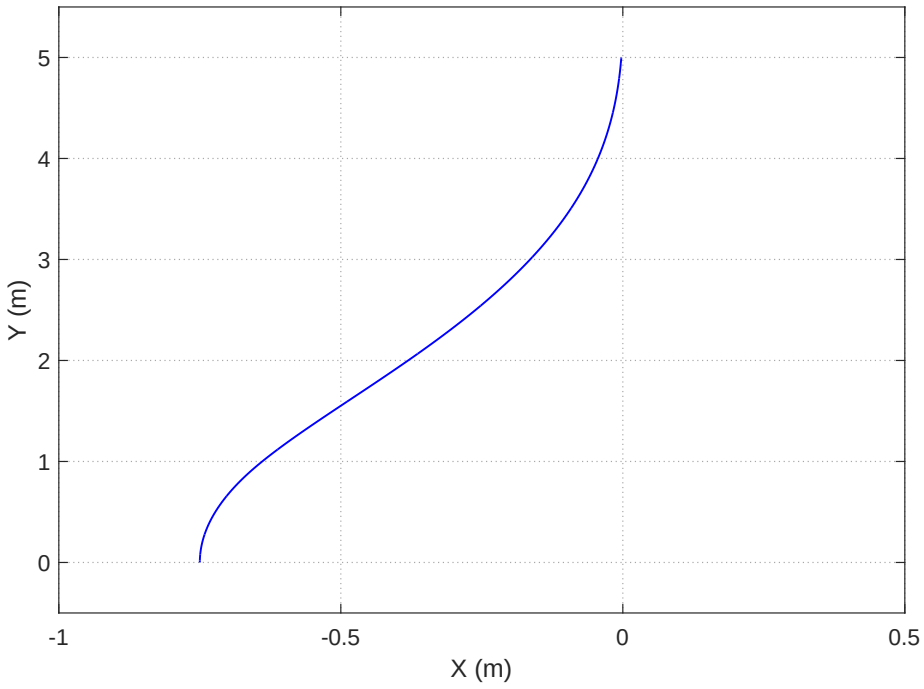


Figure 4.26: Tracking trajectory with modified control.

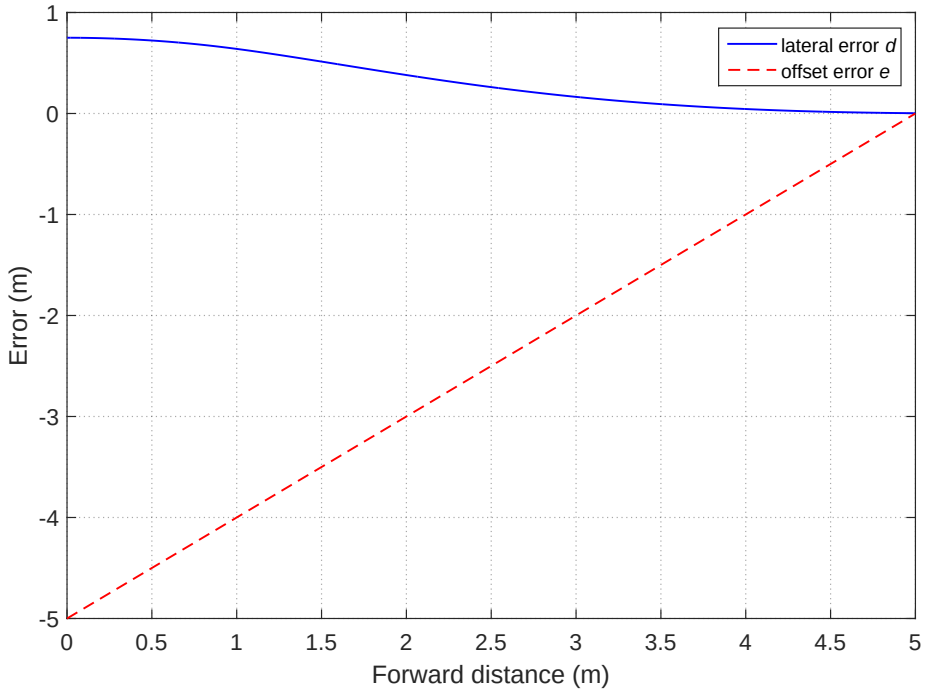


Figure 4.27: Lateral and offset errors with respect to the forward distance.

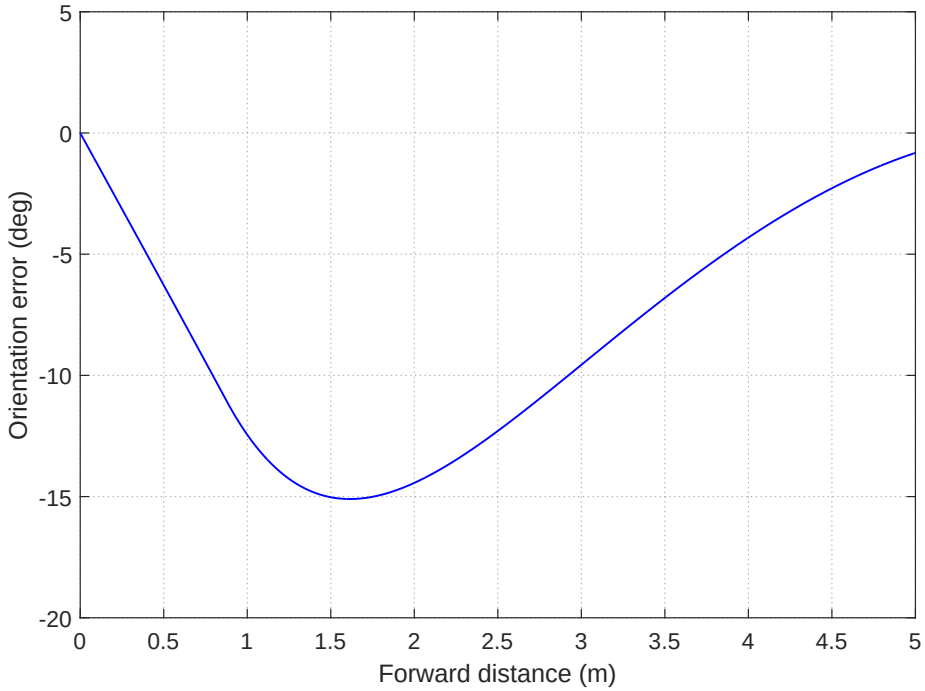


Figure 4.28: Orientation error with respect to the forward distance.

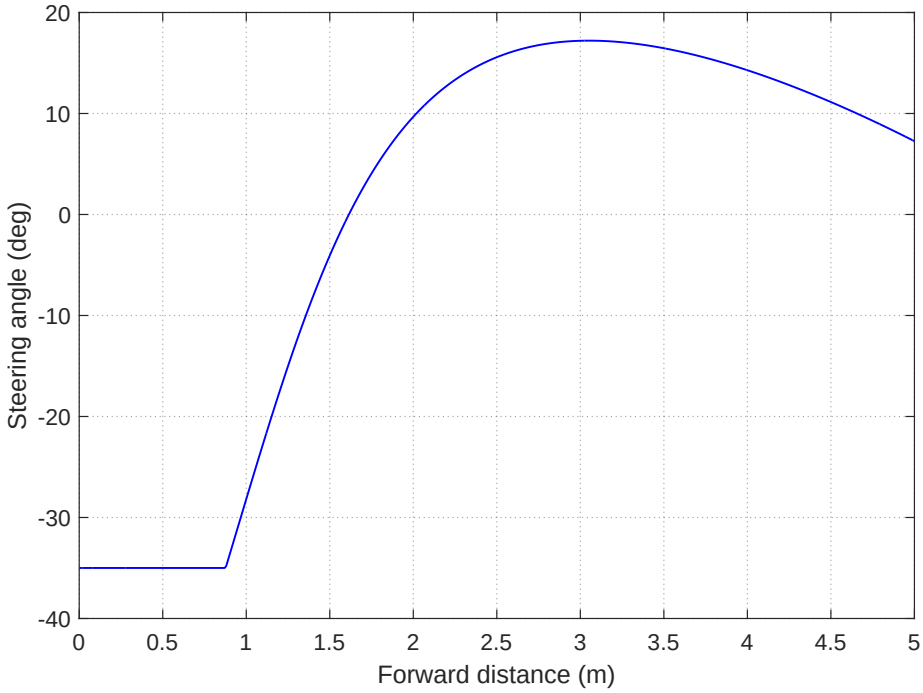


Figure 4.29: Steering angle input with respect to the forward distance.

4.2.3 Sequential point-to-point tracking

The control strategies for the single point tracking were applied in the designed control sequence. A rectangular field with two straight paths was considered and an implement width of 5 m was used. A lateral deviation of 0.5 m was set to the initial positions of both vehicles. The same velocity and steering control gains as applied in the single point tracking case were used in this simulation. The resulted tracking trajectories are shown in Figure 4.30. The initial offsets were decreased monotonically in the first tracking process and all subsequent tracking processes followed a straight line tracking.

A time sequence of the forward distances of the vehicles is shown in Figure 4.31. In the simulation, the forward tracking process took about 9 s and the implement operation was simulated as a pruning process for cranberry fields under 0.56 m/s (took 36 s). The motions of the two vehicles were perfectly synchronized since the control signals were exchanged instantly and the velocity and steering control responses were ideal and identical for the two vehicles in the simulation.

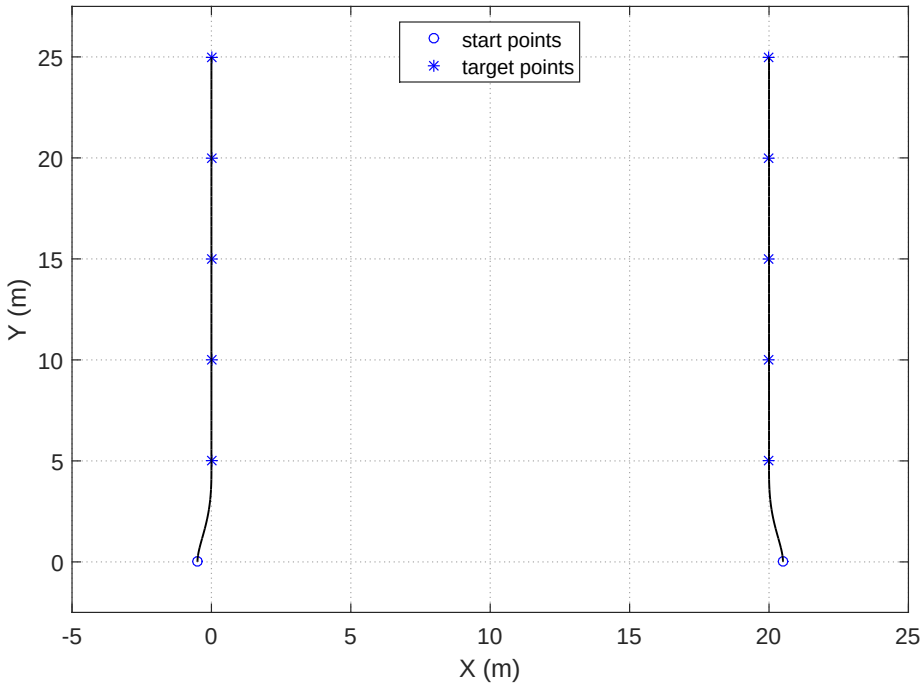


Figure 4.30: Simulated sequential point-to-point tracking trajectories.

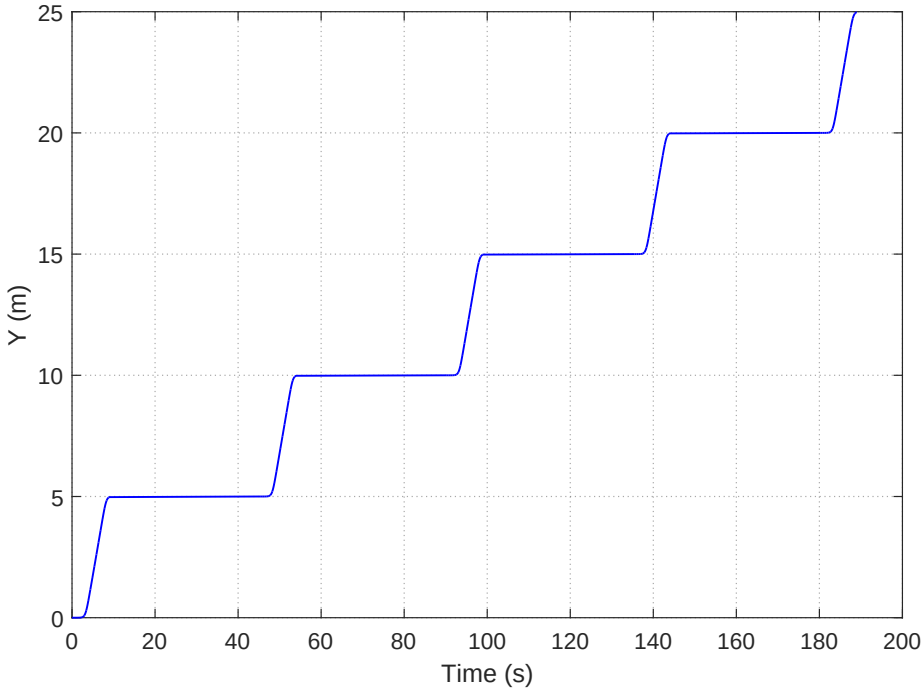


Figure 4.31: Simulated time sequence of forward distances.

4.3 Chapter summary

In this chapter, MATLAB/Simulink simulation models were built for the mobile mode and the stationary mode. The developed control models and algorithms were tested in various simulations. For the mobile mode, simulations were performed under three scenarios, the first with a constant reference velocity, the second with a varying reference velocity, and the third with curved reference paths. In all the simulations, the master and slave vehicles were able to follow the defined reference paths under the designed steering control law, and the slave vehicle could track the instantaneous positions of the master by controlling its forward velocity.

For the stationary mode, the velocity and steering angle control laws were verified in a single point tracking case, and the effectiveness of the modified control strategy was also validated. Then, the single point tracking process was repeated in a sequential point-to-point tracking case. The master and slave vehicles were able to complete the forward point tracking processes in a synchronized manner.

Chapter 5

Experimental Setup

To validate the control models and algorithms for the two operating modes in practice, a robotic experimental platform was developed and validation experiments were performed. This chapter presents the details of the developed experimental platform. The hardware and software of the guidance system as well as the experimental site and plans are introduced.

5.1 Experimental platform

The practical WSIC is a huge platform and its operation requires special attention of two to three operators at the same time. Since the purpose of the proposed experiments is to validate the designed control algorithms, it is reasonable to first develop a small-scale experimental platform to model the operation features of the WSIC and test the designed algorithms on it.

To model the practical WSIC, a scaled-down experimental platform based on two heavy-duty mobile robots was designed and constructed. Then, a group of sensors and control units, for automatic guidance and control purposes, were installed on the robots. The long truss of the practical WSIC connects the two tractors together and restricts their relative motions. However, the two tractors can be treated as independent power units as long as the distance variation between them is within the sliding range of the truss. In the modelling process, kinematic models were developed and dynamic effects of the truss were neglected under the assumption that the WSIC tractors had pure rolling and non-slipping motions. This assumption is plausible when the WSIC runs on flat and firm paths under slow speeds. Since the dynamic effects of the truss were neglected in the modelling and control design, the robotic platform was constructed without a connecting truss.

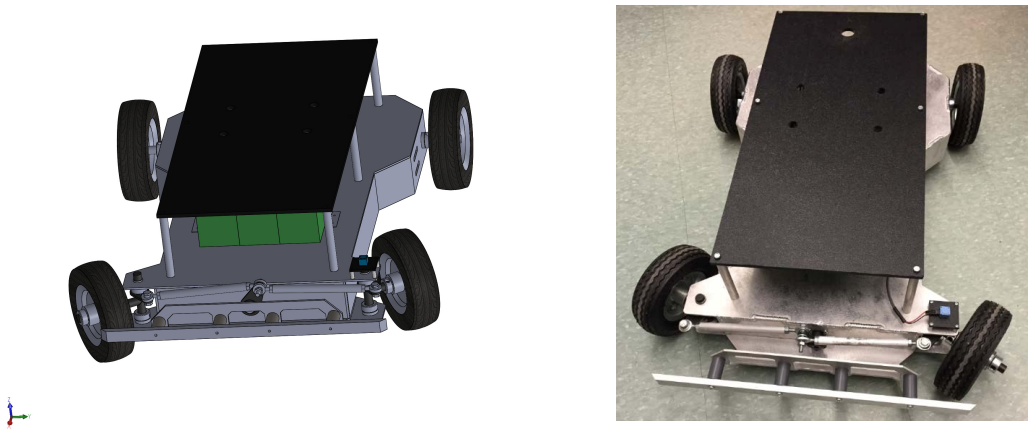
Although the experimental platform was a scaled-down model, the guidance system was designed from the consideration of a practical WSIC. Automatic guidance of the WSIC requires real-time positioning of the two vehicles with respect to their paths. Since the motions of the two vehicles need to be synchronized during the operation, their positions should be gathered and compared in real time. This requirement was conveniently fulfilled by an RTK-GPS with two rover receivers, one on each robot. The real-time orientations of the vehicles are also required for automatic guidance purposes. In this development, the orientation angles of the robots were measured by high-accuracy IMUs. Motion synchronization algorithms can be implemented when real-time communication between the two vehicles is established. Considering the wide distance between the two WSIC vehicles, a point-to-point wireless network was built using two XBee modules for the experimental platform.

5.1.1 Hardware

The main hardware used for the experimental platform includes two car-like mobile robots, a dual-rover RTK-GPS for real-time positioning, two IMUs for robot orientation tracking, two XBee-PRO modules for wireless communication, and several processors for running control algorithms and interfacing with the guidance sensors and communication units.

(1) Mobile robots

The experimental WSIC platform is a scaled-down model of the real WSIC machine. Two car-like mobile robots capable of rear-wheel driving and front-wheel steering were used to model the control behaviours of the two tractors. The two robots were customized and manufactured by SuperDroid Robots Inc. Both robots have four 6-inch diameter wheels, a total width of 54.4 cm and a total length of 70.8 cm. The aluminum body frames of the robots were well built and welded, which made them strong enough to support the weight of all necessary equipment. Figure 5.1 shows the 3D model of the designed mobile robot and the real one after manufacturing and assembly. On each robot, two 24V DC gear motors were used to drive the rear wheels and an additional 24V DC gear motor was used to provide steering to the front wheels. The front wheels were installed and adjusted to be able to make turns within a range of about $\pm 32^{\circ}$. A Honeywell 308N5K rotary potentiometer was mounted on the steering shaft of the left wheel to sense the steering angles. The mechanical specifications of the robots are listed in Table 5.1. Each robot used three 12V 9.6 Ahr LiFePo battery packs (green packs in Figure 5.1(a)) to provide power to the motors and all other necessary hardware.



(a) (b)

Figure 5.1: (a) 3D model and (b) photo of the mobile robots.

Table 5.1: Parameters of mobile robots.

Description	Value
Wheelbase	49.5 cm
Tread width	48.0 cm
Wheel diameter	15.24 cm
Steering range	±32°
Speed range	±1.2 m/s
Weight	12 kg
Carry capacity	18 kg

(2) RTK-GPS

The RTK-GPS used for the experimental platform mainly consisted of three NovAtel GPS receivers, one working as the base station and the other two as rovers. The models of the receivers and other relevant equipment, including the antennas and radios, are provided in Table 5.2. The dual-rover RTK-GPS used one DL-V3 receiver as the base station and two FlexPak6 receivers as rovers. Three 900 MHz wireless radio modems were configured to work with the receivers to establish the real-time data transfer of the differential correction messages. The base station was set up accurately at a local point whose absolute position was adequately searched and perfectly known. The GPS antennas of the rovers were mounted on the top of the robots, straight up above the centre of the rear axle. This mounting point was deemed as the real-time control position of the robot during the run. The rover receivers output positioning messages at a rate of 10 Hz with a reported relative horizontal position accuracy of 2 cm. They were also used to measure robot velocities with a reported accuracy of 0.03 m/s.

Table 5.2: RTK-GPS equipment.

Equipment	Model	Quantity	Brand
Base receiver	DL-V3	1	NovAtel
Rover receiver	FlexPak6	2	NovAtel
Base antenna	GPS-702-GG	1	NovAtel
Rover antenna	GPS-600-LB	1	NovAtel
Rover antenna	G5Ant-3AT1	1	Antcom
Radio modem	n920-ENC	3	Microhard

(3) IMUs

Two miniature IMUs were installed on the robots to measure their real-time motion states, one on each robot. The model of the IMUs is 3DM-GX3-25, manufactured by LORD MicroStrain company. A picture of the IMU is shown in Figure 5.2. The 3DM-GX3-25 combines a triaxial accelerometer, a triaxial gyroscope, a triaxial magnetometer, temperature sensors, and an onboard processor running a sensor fusion algorithm to provide accurate static and dynamic inertial measurements. The IMUs can measure dynamic orientation Euler angles (pitch, roll, and heading) with 2° accuracy and less than 0.1° resolution, and accelerations with 0.04 mg in-run bias stability and 0.002 g initial bias error.

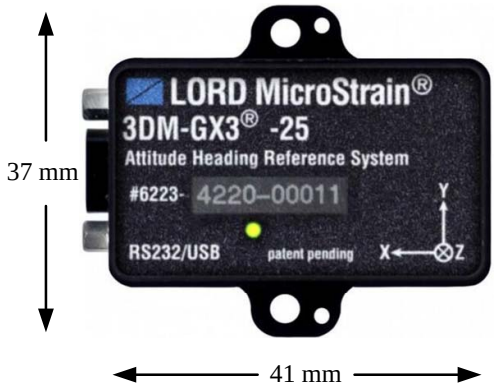


Figure 5.2: A picture of the 3DM-GX3-25 IMU.

(4) XBee modules

To design an automatic guidance system for the WSIC requires real-time data communication between the two vehicles. This is preferably realized through wireless communication considering the wide distance between the two vehicles. Two XBee-PRO S2B wireless modules from Digi International Inc. were used on the experimental platform to set up a

point-to-point wireless network. The XBee modules operate at 2.4 GHz of the ISM radio band and can reach a theoretical data rate of 250 kbps. The outdoor line-of-sight range can reach up to 3.2 km. The large band width and far application range are sufficient for the transmission of control data between the two WSIC vehicles. A picture of the XBee module within a USB interface board is shown in Figure 5.3.

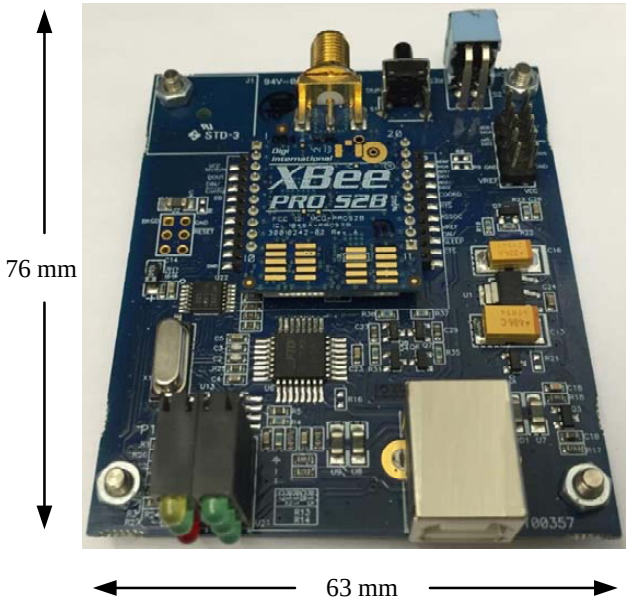


Figure 5.3: A picture of the XBee module within a USB interface board.

(5) Processors

Several processors were used for the experimental platform. On each robot, a laptop computer, an Arduino Mega ADK microcontroller, and a Sabertooth dual 12A motor driver were installed.

A Dell laptop was used as the central control unit to communicate with the sensors and other processors and run the developed control algorithms. The Dell laptop has an Intel Core-i7 2.7 GHz processor and 8 GB memory and runs Windows 7 operating system. A series of communication and control programs were developed in the computers.

An Arduino Mega ADK microcontroller was used to communicate with the onboard laptop and the motor driver. The Mega ADK (Figure 5.4) has a 16 MHz ATMega2560 processor with 8 KB RAM and 256 KB flash program memory. It provides multiple input and output capabilities including 54 digital input/output pins, 16 analog input pins, 4 TTL serial communication ports, one USB port, and one SPI communication port. The Mega

ADK receives the desired velocity and steering angle values from the onboard computer, and sends proper voltage control signals to the motor driver through two digital output pins. The Mega ADK has 16 analog inputs, each of which provides 10 bits of ADC resolution (i.e. 1024 different values). The board used one analog pin to sense the potentiometer position and calculate the steering angle based on that.

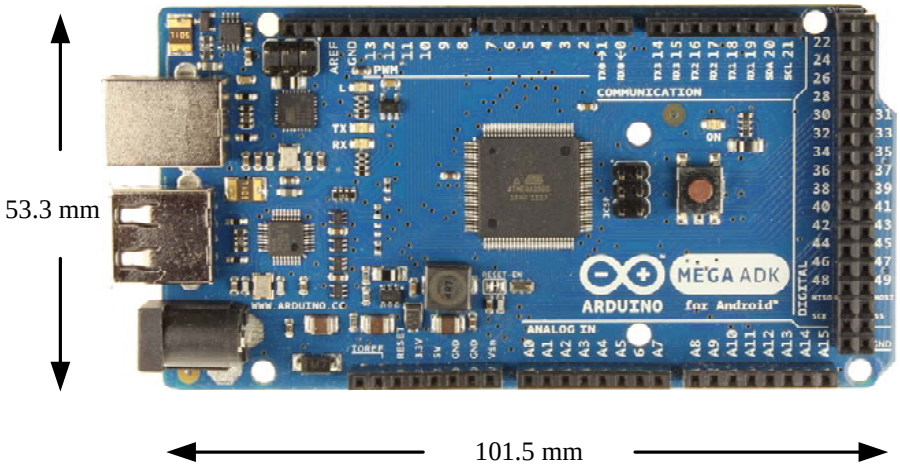


Figure 5.4: A picture of the Arduino Mega ADK microcontroller.

The Sabertooth dual 12A motor driver (Figure 5.5), manufactured by Dimension Engineering company, is an efficient and easy-to-use dual-channel motor driver suitable for medium powered robots up to 45 kg. One channel of the driver was connected to the driving motors and the other channel was connected to the steering motor. The driver receives motor control commands from the Arduino Mega board and converts them into corresponding voltages to drive the motors. Simplified serial operating mode was used for the motor driver. A TTL level 8N1 serial data stream was connected to the terminal S1 on the motor driver, and the control was with single byte commands.

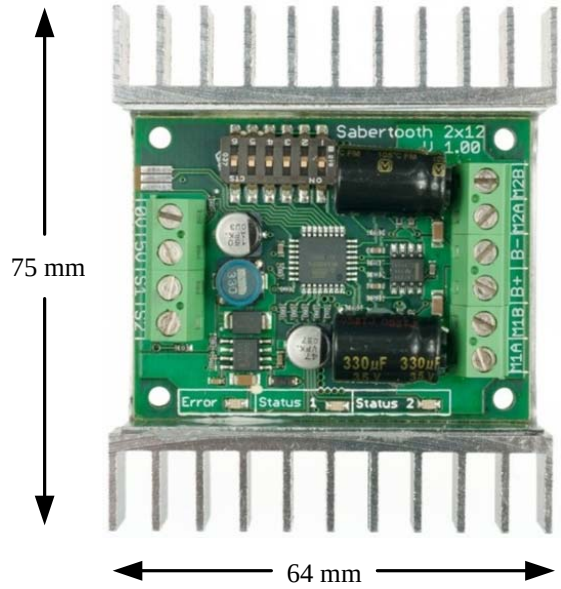


Figure 5.5: A picture of the Sabertooth dual 12A motor driver.

5.1.2 Software

The main software for communications between the onboard host computer and the peripheral hardware was developed in Windows 7 system using MATLAB 2015a. The MATLAB environment provides a series of serial port interfacing functions for the host computer to communicate with devices connected by serial ports. A MATLAB program was written and tested for the following tasks: reading and parsing the GPS output sentences, reading and parsing the IMU output strings, sending and receiving data through the XBee module, sending and receiving data through the Arduino board, and processing the control algorithms. A complete listing of the program codes can be referred to in Appendix A.

(1) Communications

Several subprograms were developed for the communications between the onboard host computer and the GPS, IMU, XBee, and Arduino.

The NovAtel GPS receiver installed on the robot communicates with the onboard computer through a 9-pin RS232 serial port. The receiver was configured to output standard NMEA 0183 sentences (GPGGA and GPRMC) at a frequency of 10 Hz. These sentences incorporate a series of guidance information of the vehicle. The MATLAB program parses these sentences and extracts the key positioning data including altitudes, latitudes, longitudes, and speeds. To use the data in the control algorithms, a local origin with

pre-measured latitude, longitude, and altitude was set, and the moving latitudes and longitudes of the vehicles were converted into local east-north-up (ENU) coordinates. This conversion was realized by the MATLAB function `geodetic2enu`. For convenience, the origin was set at the location of the base station.

The 3DM-GX3-25 IMU outputs its real-time measurements to the onboard computer through a 9-pin RS232 to USB converted port. The raw packets of the sensor output are formatted hexadecimal strings which combine a repeating packet header, a data field, and a checksum. A MATLAB program was written to parse the output packets and convert the data field into decimal angle values. The IMU was configured to output orientation angles at a frequency of 20 Hz.

The XBee module works as a serial port device and communicates with the onboard computer through a USB port. The XBee pair were configured to work in transparent mode, one as the coordinator and the other as the router. A private point-to-point wireless network was established between the two robots and data were transmitted and received over the air. The serial communication parameters for the XBee modules were configured by MATLAB programs. At the transmitting end, the program packetizes the data to be transferred and sends them to the XBee module; at the receiving end, another program scans the XBee module and parses the received data packets.

The Arduino microcontroller communicates with the onboard computer through another USB serial port. The onboard computer works as a host and runs upper-level MATLAB programs to process control algorithms. An Arduino program was written to parse the control commands sent from the host and control the driving and steering motors. The lower-level Arduino program runs a proportional-integral-derivative (PID) algorithm to control the steering motor.

(2) Control system structure

The whole control system was developed at two levels, as shown in Figure 5.6. At the upper level, the control system was developed to read the defined reference information, parse sensor measurements, process algorithms, and generate control signals including velocities and steering angles. At the lower level, the motor control processors were programmed to receive the signals and control the driving and steering motors accordingly.

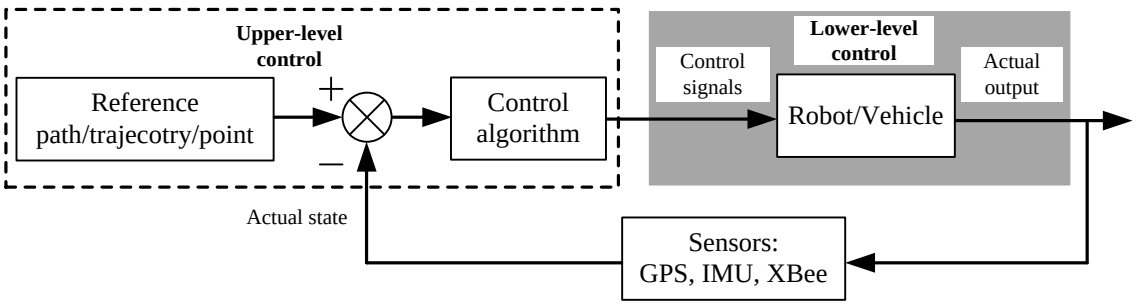


Figure 5.6: Structure of the control system.

5.1.3 Integration

The hardware used on both robots has the same integration structure. The onboard computer serves as the central control unit to interface with all the sensors and other processors. It provides multiple USB and serial ports for communications with the GPS, IMU, XBee, and Arduino. The structure of the hardware interfacing is shown in Figure 5.7.

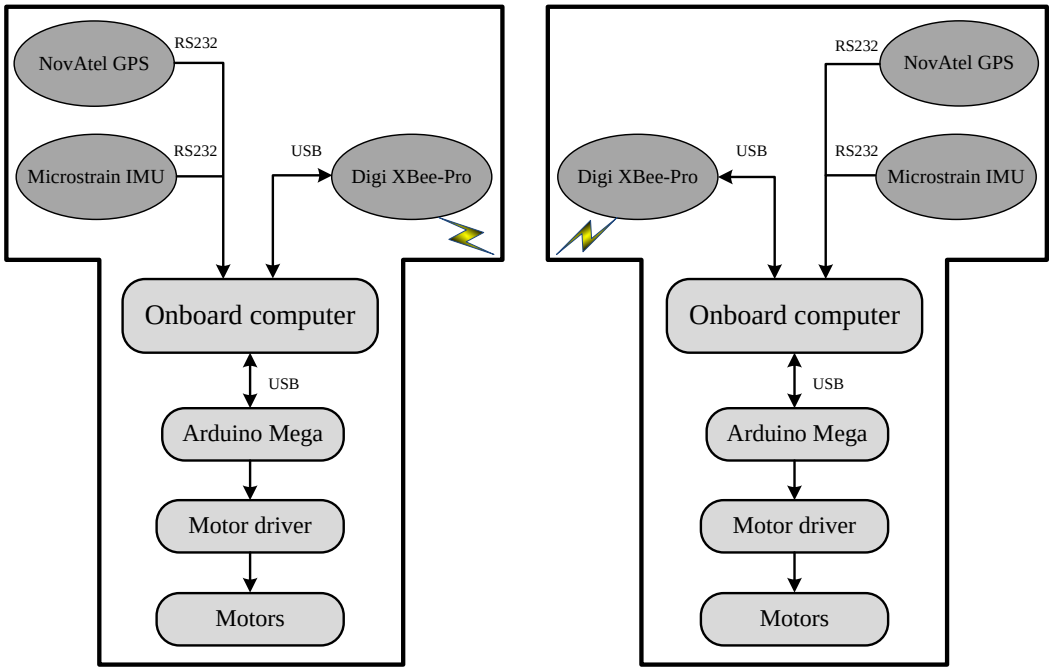


Figure 5.7: Structure of the hardware integration.

The onboard computer reads the measurements from the GPS and IMU, processes the control algorithms, and sends the motor control commands to the Arduino board. The data communication between the two robots, which is required for the cooperative

synchronization control, is established wirelessly by the two XBee modules. After installing all the hardware on the robots, the developed experimental WSIC platform is shown in Figure 5.8.

Figure 5.8: A picture of the experimental WSIC platform.

5.1.4 Calibrations

The motions of the mobile robots are controlled by their velocities and steering angles. After the robots were constructed and assembled, the driving and steering mechanisms were calibrated and their performances were tested.

(1) Driving

Two 24V DC gear motors were mounted on the rear axle to provide driving for the robots, one connected to the left wheel and the other to the right wheel. The power inputs of the two motors were connected together and controlled by one channel of the Sabertooth motor driver. To make sure that the motors output the same rotating speed, their voltage/rpm responses were calibrated. As shown in Figure 5.9, both motors had good voltage/rpm linearity and their responses were nearly identical.

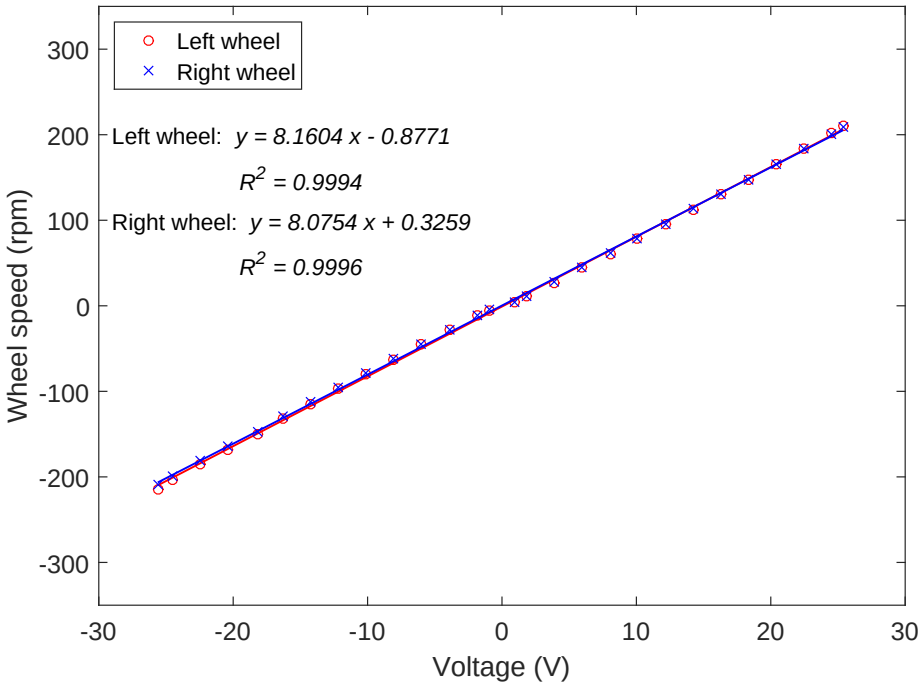


Figure 5.9: Voltage/rpm calibration of the driving motors.

The driving velocity is controlled by the motor driver output power (8 bit, -127 to 127). A linear calibration relating the driving power to the wheel rotating speed was completed. Figure 5.10 shows the velocity measurements using the onboard GPS receiver when the robot was commanded to run at 0.4, 0.8, and 1.2 m/s on a firm and level ground. The average steady state error was within 0.1 m/s, and this value was set as the control interval for the velocities.

(2) Steering

The steering of the robots is controlled by an additional 24V DC gear motor. The front wheels were connected to a rocker arm-rod linkage. The motor was mounted in the middle of the front axle, connecting both rods together. The positions of the rocker arms and lengths of the rods were adjusted so that the wheels could make a maximum 32° turning angle, both to the left and to the right. The Honeywell potentiometer was mounted coaxially on top of the left wheel rocker arm to detect the magnitude of the steering. In control design of automatic guidance, a four-wheel tractor is usually simplified as a two-wheel bicycle model and the front wheels are represented by one virtual wheel located in the middle of the front axis. The relation of the virtual wheel steering angle δ and the left wheel steering angle δ_l of the robots is shown in Figure 5.11. Ideally, when the vehicle is

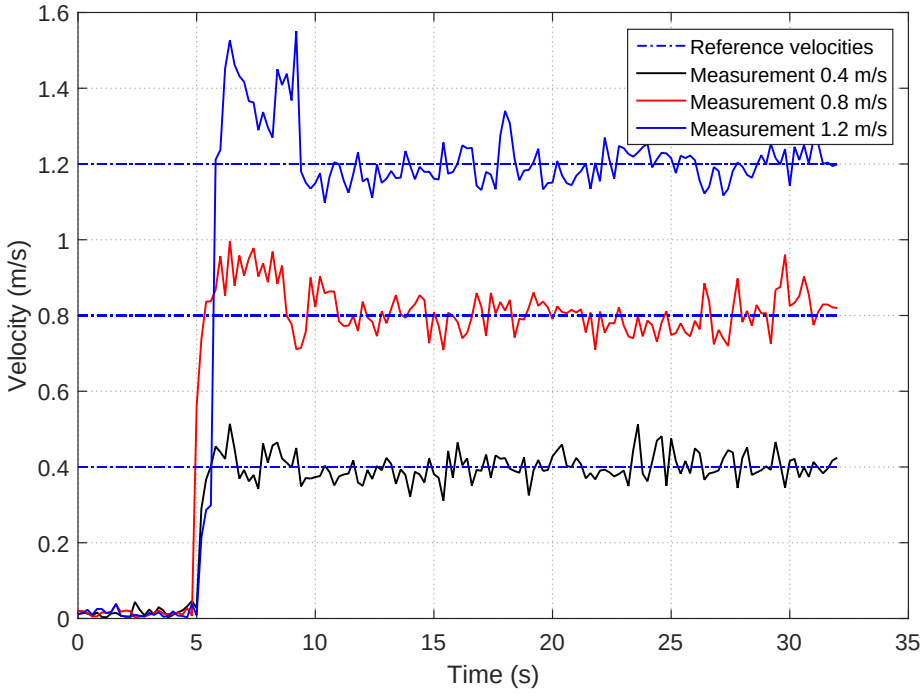


Figure 5.10: Robot velocity tracking results.

turning without slippage, the instantaneous rotating centres of the left, right, and virtual wheels would meet at a single point c, locating on the extended rear axis. In the figure, l and b stand for the vehicle wheelbase and tread width respectively. The relation of δ and δ_l can be represented by the following equation

$$\delta_l = \begin{cases} \arctan \frac{l \tan \delta}{l - 0.5b \tan \delta}, \delta \geq 0 \\ \\ \arctan \frac{l \tan \delta}{l + 0.5b \tan \delta}, \delta < 0 \end{cases} \tag{5.1}$$

This equation was used in the upper-level control program for converting the desired steering angle calculated by the control algorithms to the actual steering angle that should be sent to the lower-level control program.

The steering potentiometer was used to measure the true left wheel steering angles. To convert the potentiometer reading value to an angle, the following calibration procedures were finished: (1) place the robots on a level ground, turn the wheels to the maximum left, maximum right, and neutral positions, and draw lines on the ground to represent the wheel directions; (2) record potentiometer readings when the wheels were at the above positions; (3) measure the exact angles using a protractor; (4) calculate the equation relating the potentiometer readings to the angle values. The following two equations were obtained for

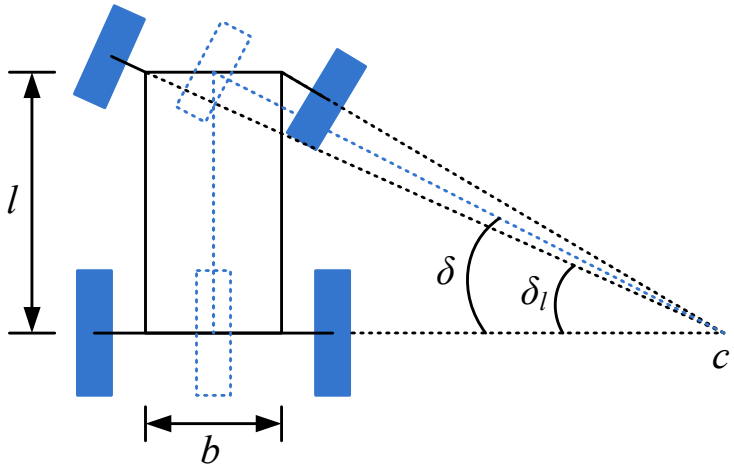


Figure 5.11: Relation of the virtual wheel and left wheel steering angles.

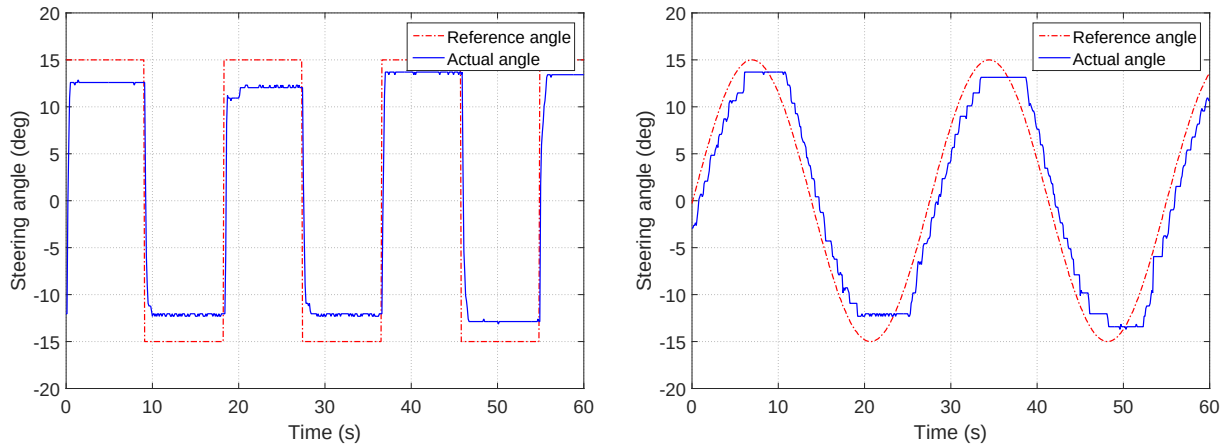
the robots

$$y = 0.26446x - 154.446 \tag{5.2}$$

$$y = 0.26506x - 147.506 \tag{5.3}$$

where y is the steering angle in degrees, and x is the potentiometer reading, which ranges from 463 to 705 in equation (5.2) and from 432 to 681 in equation (5.3).

The performance of the steering mechanism was tested to determine the control accuracy of the steering angle. Figure 5.12 shows the steering tracking results for two reference signals, one square wave signal and one sine wave signal. The PID parameters in the tests were set to $k_p = 1.5, k_i = 0.25$, and $k_d = 0.05$ so that the motor did not generate overshoots. As the results show, the maximum steering tracking error was about 4° and the steering delay was around 2 s. The steering error and delay were mainly due to the reaction torque that the motor had to overcome and the backlash that existed in the motor/wheels linkage. Since there was no assisted-steering mechanism between the motor output shaft and the wheels, the motor had to conquer large reaction torque generated by the weight and load of the robot. The linkage between the motor output shaft and the wheels also had a backlash of about 1.1° that influenced the overall control accuracy. To prevent the steering motor from constantly changing directions and jittering, which are detrimental to its life span, the motor was set not to response to steering angle errors of less than 3° in the control program.



(a) (b)

Figure 5.12: Robot steering angle tracking results.

5.2 Experimental methods

5.2.1 Experimental site

The experimental site needed to meet two conditions: a flat and firm surface for the robotic platform to run on and good accessibility to the GPS satellites signals. The site was chosen to be on the top floor of an outdoor parking garage, which met the above two requirements.

As shown in Figure 5.13, two parallel straight paths, spaced 6.5 m, were designed on the surface for the experimental platform, path 1 for the master robot and path 2 for the slave robot. The RTK-GPS receivers installed on the robots output absolute positioning messages including latitudes, longitudes, and altitudes. To convert these messages into local Cartesian coordinates, a local origin with pre-measured latitude, longitude, and altitude was set, and the conversion was processed by the MATLAB function `geodetic2enu`. The geodetic coordinates of measured points were converted into (x, y, z) coordinates in a local ENU Cartesian system. Based on that, the equations of the parallel paths could be determined.

Figure 5.14 shows the experimental site with designed target points for the stationary mode. Once the equations of the parallel paths were determined, the coordinates of the target points could be easily calculated and located on the paths.

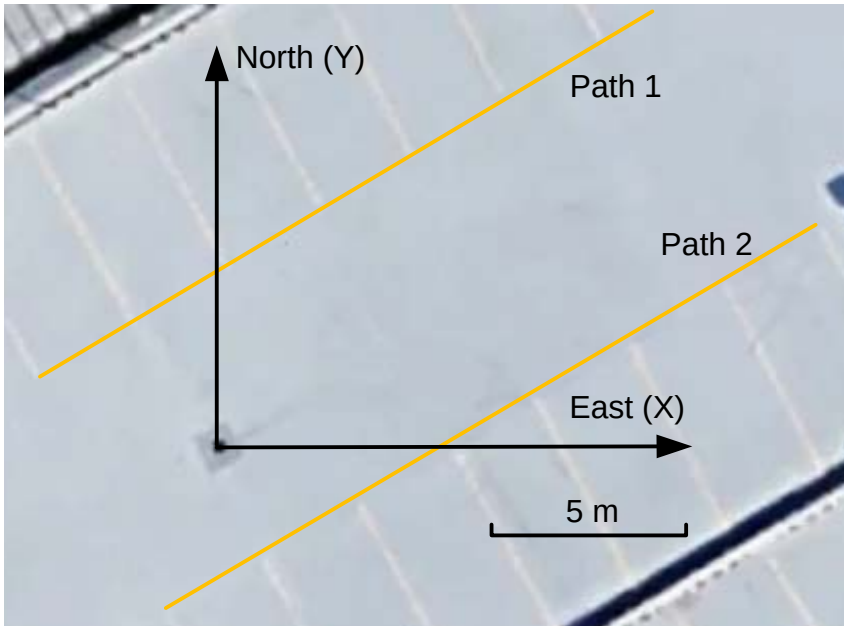


Figure 5.13: Experimental site with designed parallel paths for the mobile mode.

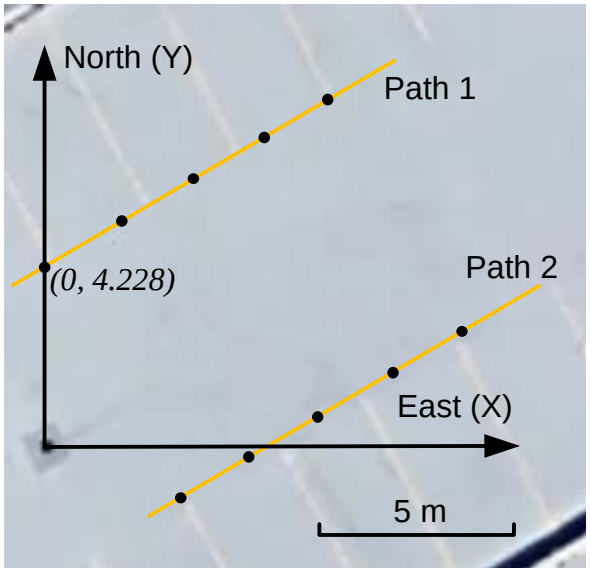


Figure 5.14: Experimental site with designed target points for the stationary mode.

5.2.2 Experimental plan for the mobile mode

The validation experiments for the mobile mode were planned in two steps and two series of tests were performed. First, the path following control for one robot was tested. The designed steering control law was applied in the experiments. In this step, different groups of control gain values were tested and proper ones were determined. In the second step,

the synchronous tracking control of two robots was tested and the overall performance of the guidance system was evaluated.

Since the steering mechanism of the experimental robots had an error around 4° and a response delay near 2 s, the maximum velocity that the robots should run at was tested and determined before the formal validation experiments. From the tests, when the robot velocity exceeded 0.7 m/s, the current steering system could not respond fast enough to correct the increasing lateral deviations. Through a theoretical analysis, this was due to the small wheelbase of the robots, which is only 0.495 m. From the vehicle model equation (3.6), the length of the wheelbase l has a significant influence on the resulted vehicle position. This can be seen by applying the same forward velocity to two vehicles that have the same steering capability (4° error and 2 s delay) and different wheelbase lengths, 0.495 m for a mobile robot and 3 m for a normal tractor. For the 0.495-m robot, a 0.8 m/s forward velocity will result in 0.18 m lateral error and 12.95° orientation error after 2 s, whereas for the 3-m tractor, the errors will be only 0.03 m and 2.14°. Based on the tests and analysis, the maximum velocity input for the robots in the experiments was set to 0.6 m/s.

For the path following experiments, three different scenarios were considered: constant velocity with zero initial deviation, varying velocity with zero initial deviation, and constant velocity with large initial deviation. The first two scenarios were to test the adaptability of the steering control law to different reference velocities, and the third one aimed to verify the capability of the steering control law to recover the robot from large initial deviations. After the single-robot path following control was validated, the synchronous tracking control for two robots was tested, and the experiments of two scenarios were performed, one with a constant reference velocity and one with a varying reference velocity.

5.2.3 Experimental plan for the stationary mode

The validation experiments for the stationary mode were planned in two steps as well and two series of tests were performed. The first series considered a single point tracking case, which aimed to verify the effectiveness of the velocity and steering angle control laws. In this step, different groups of control gain values were tested and proper ones were determined. The second series executed the parallel point-to-point tracking control sequence, which tested the overall performance of the periodic point-to-point tracking control and the reliability and efficiency of the whole system.

Considering the strong stiffness and the backlash that existed in the physical steering mechanism, the maximum velocity that the robots should run at was experimentally tested.

After the trial runs, the maximum velocity for the stationary mode experiments were set to 0.4 m/s. For the single point tracking case, two implement widths, 2-m and 3-m, were considered. The modified point tracking strategy was applied, and the virtual target was located 1 m further to the actual one. In the sequential point-to-point tracking experiments, the single point tracking process was repeated in the control sequence. The motions of the master and slave robots were recorded and compared to analyze their synchronization status during the operation.

5.3 Chapter summary

In this chapter, the experimental setup for the validation experiments was introduced. A scaled-down WSIC experimental platform was built based on two heavy-duty mobile robots. The hardware and software of the platform were introduced. The robots were instrumented with a dual-rover RTK-GPS, IMUs, XBee-PRO wireless communication modules, and a group of control processors. Interfacing and communication programs were developed in the MATLAB environment in the host computer, and the control structure was developed at two levels: upper level for control algorithms calculation and lower level for robot velocity and steering angle control. The driving and steering capabilities of the robots were calibrated and tested.

To perform the validation experiments, a proper site located on the top floor of an outdoor parking garage was chosen. The site met the two important requirements for the experiments: a flat and firm surface and good accessibility to the GPS satellite signals. The parallel straight paths and the series of target points, required respectively for the mobile mode and stationary mode tests, were designed. Lastly, the experimental plans for both operating modes were introduced.

Chapter 6

Results and Discussion

6.1 Mobile mode

6.1.1 Path following

The first series of tests aimed to verify the effectiveness of the steering control law (equation (3.12)) for the path following task. Several preliminary experimental tests were performed first to tune and determine the control gains for the steering control law. After the tests, k_1 and k_2 in equation (3.12) were set to 2.4 and 1.4 respectively.

(1) Constant velocity with zero initial deviation

In this test, the robot was placed on the reference path with a near zero initial deviation and a constant reference velocity 0.3 m/s was applied. The recorded robot trajectory and the reference path are plotted in Figure 6.1. As shown in the figure, the robot followed the desired path with small deviations. The variations of the lateral and orientation errors are shown in Figures 6.2 and 6.3 respectively. The figures show that the lateral errors were restricted within a 0.06-m range and the orientation errors were kept less than 4° during the run. The reference steering angles and the actual robot steering angles are compared in Figure 6.4. It is noticeable that when the lateral and orientation errors varied around zero, the steering angle inputs also fluctuated within a few degrees around zero. The actual steering of the robot delayed the reference commands by a time amount about 2 s and reduced the reference magnitudes by about 3°.

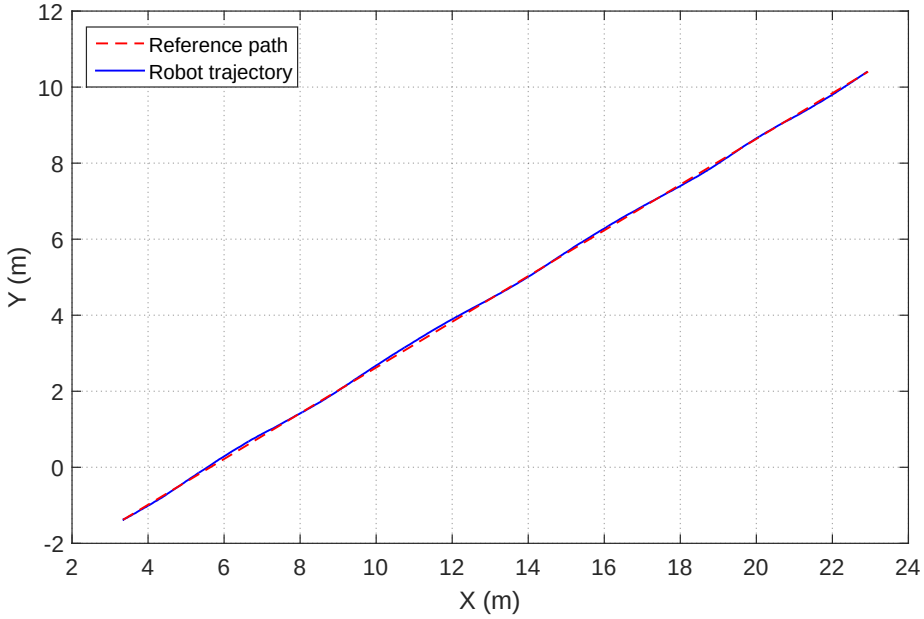


Figure 6.1: Path following trajectory, constant velocity with zero initial deviation.

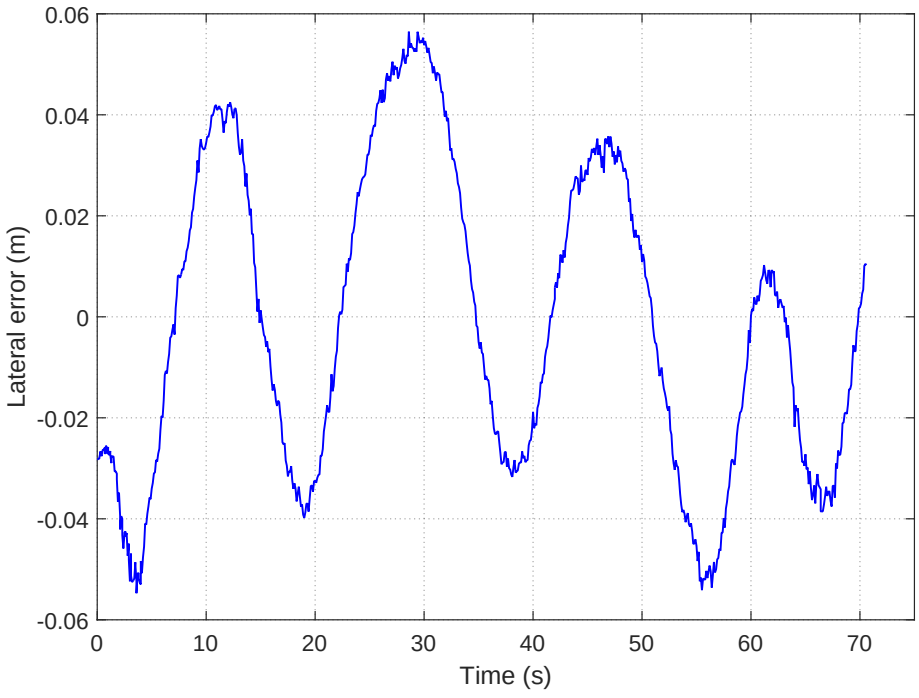


Figure 6.2: Lateral error, constant velocity with zero initial deviation.

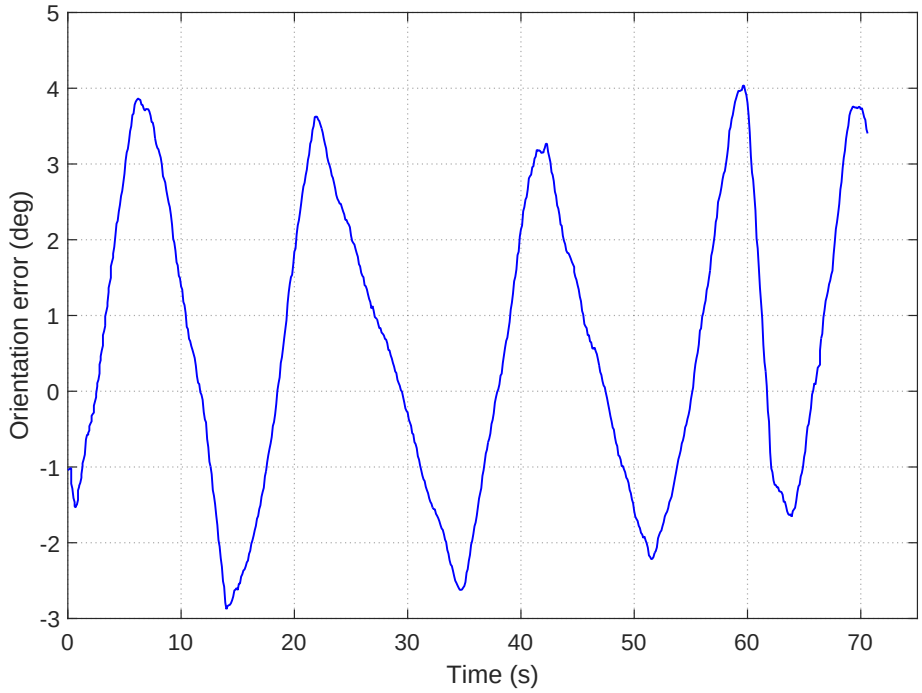


Figure 6.3: Orientation error, constant velocity with zero initial deviation.

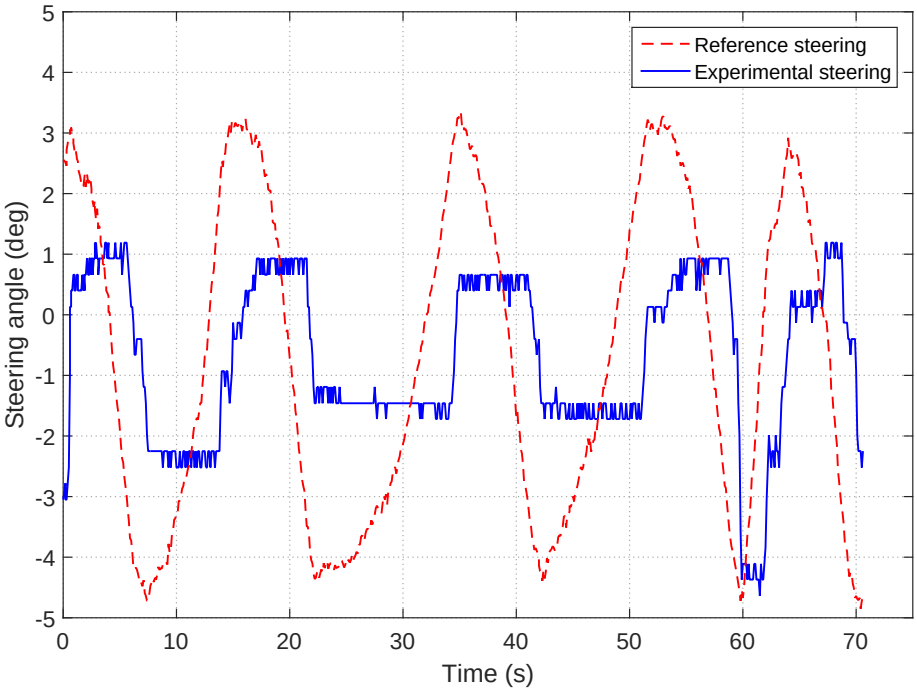


Figure 6.4: Steering angle input, constant velocity with zero initial deviation.

(2) Varying velocity with zero initial deviation

The designed steering control law is velocity independent. To test the system response to varying velocities, the experiment with a velocity input ranging from 0.2 to 0.6 m/s was performed. The sinusoidal velocity input is shown in Figure 6.5, and the resulted robot trajectory is shown in Figure 6.6. The variations of the lateral and orientation errors are shown in Figures 6.7 and 6.8 respectively. It can be seen that the maximum lateral error was less than 0.08 m and the maximum orientation error was less than 5°. The steering angle input is displayed in Figure 6.9. The performance of this test is similar to that of the constant velocity test.

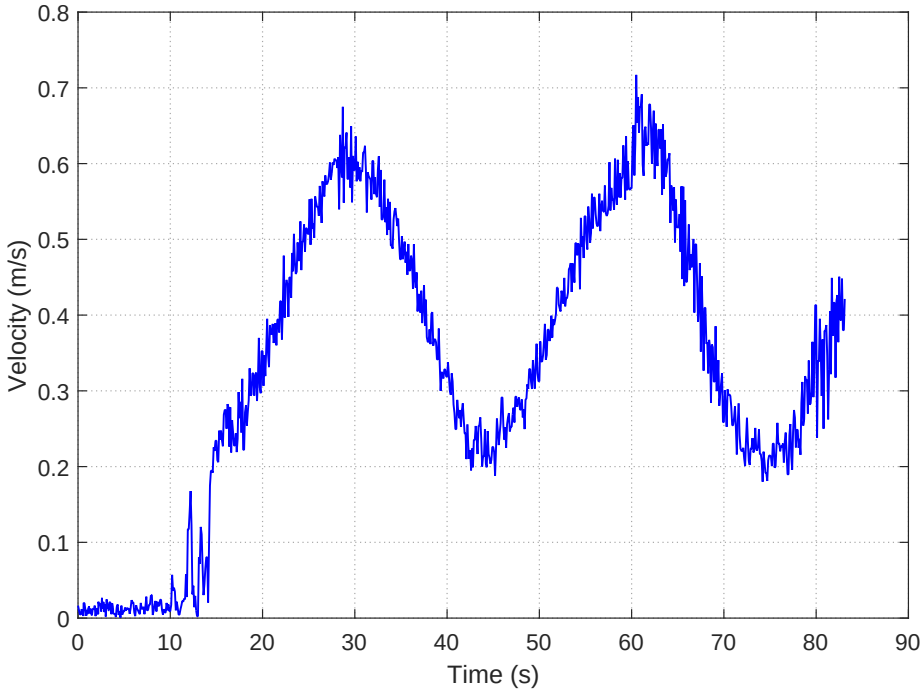


Figure 6.5: Varying velocity input.

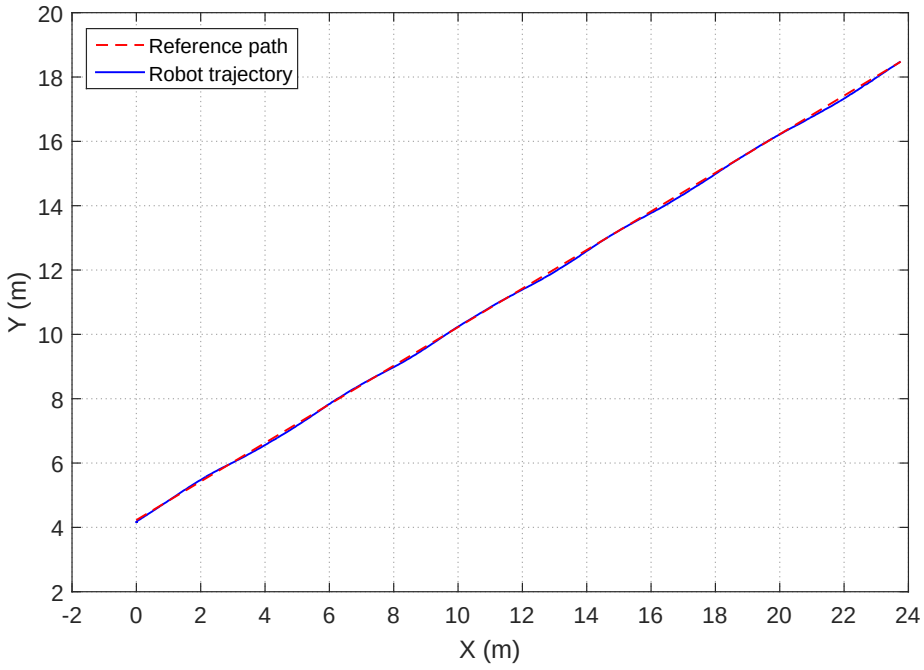


Figure 6.6: Path following trajectory, varying velocity with zero initial deviation.

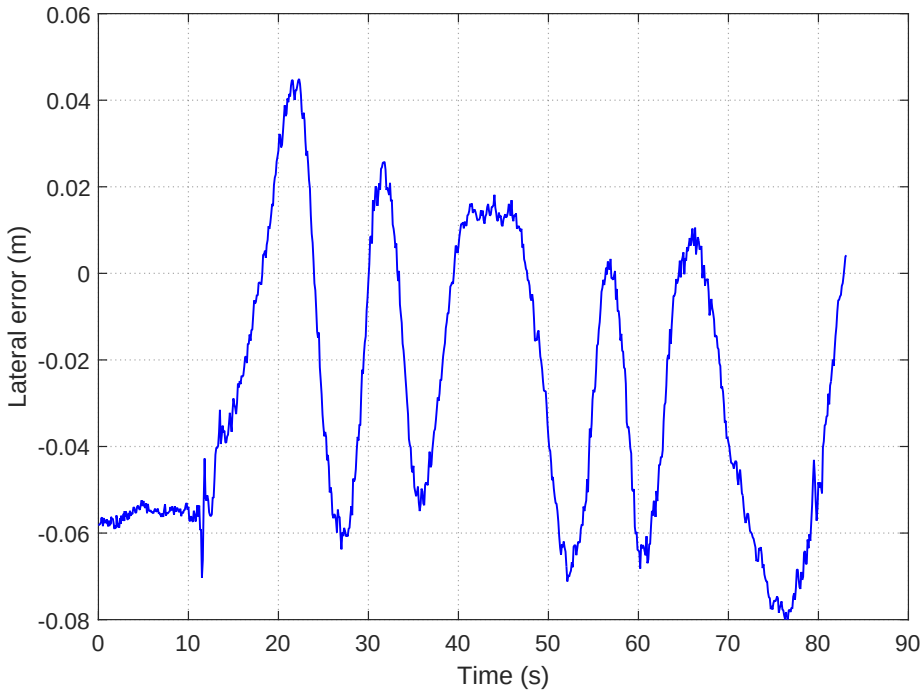


Figure 6.7: Lateral error, varying velocity with zero initial deviation.

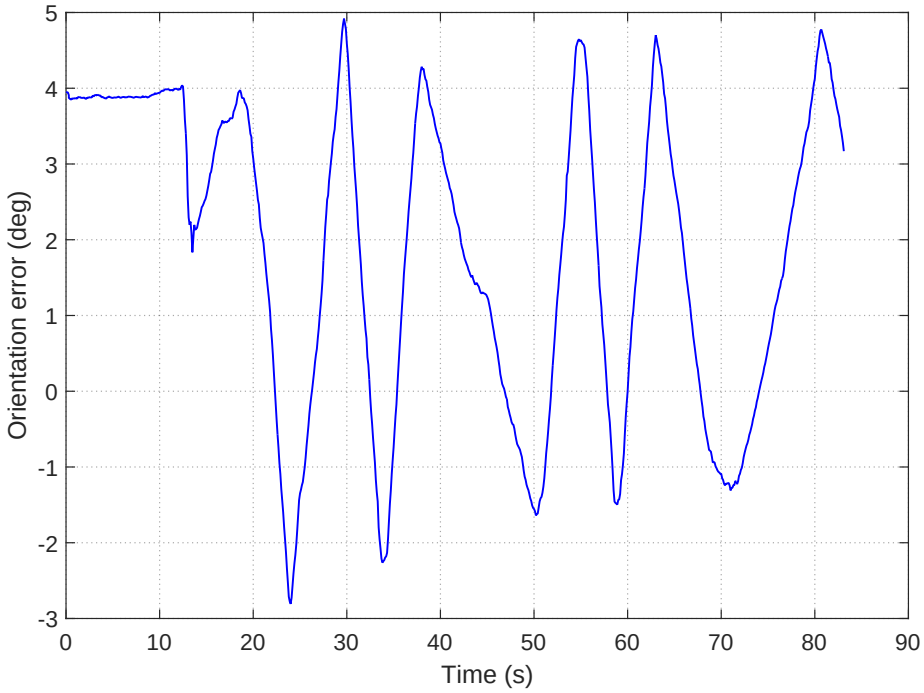


Figure 6.8: Orientation error, varying velocity with zero initial deviation.

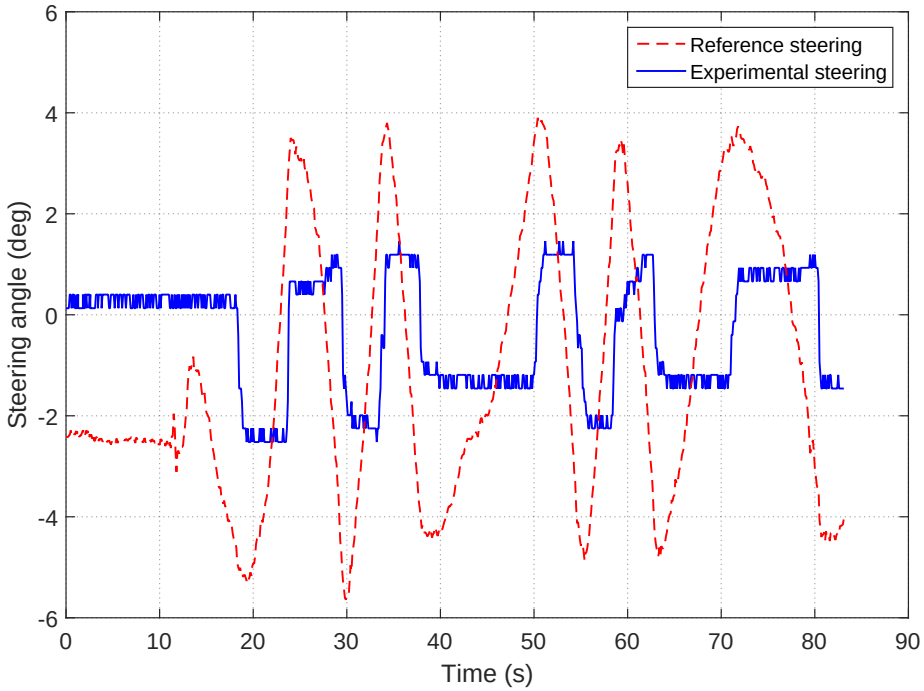


Figure 6.9: Steering angle input, varying velocity with zero initial deviation.

The statistical errors of the two experiments are presented in Table 6.1. The average and RMS lateral errors were 0 and 0.03 m for the constant v experiment, and -0.03 m and 0.04 m for the varying v experiment. The maximum and minimum lateral errors were 0.06 m and -0.05 m for the constant v experiment, and 0.04 m and -0.08 m for the varying v experiment. The average and RMS of the orientation errors were 0.59° and 2.02° for the constant v experiment, and 1.91° and 2.78° for the varying v experiment. For both tests, the largest orientation error was less than 5°. It can be concluded that with the current steering mechanism on the robot, the designed steering control law can make the robot follow a reference path on a flat and firm ground with less than 0.1 m lateral error and less than 5° orientation error.

Table 6.1: Path following errors.

	constant v 0.3 m/s		varying v 0.2-0.6 m/s	
	d (m)	θ_p (°)	d (m)	θ_p (°)
Average	0	0.59	-0.03	1.91
RMS	0.03	2.02	0.04	2.78
Max	0.06	4.03	0.04	4.92
Min	-0.05	-2.88	-0.08	-2.81

(3) Constant velocity with large initial deviation

In the above tests, the robot was placed on the reference path with near zero initial errors. To test the ability of the steering control law to correct the robot from large initial deviations, the experiment when the robot was placed with a large initial error was performed. The control gains were kept unchanged and the reference velocity was set to 0.4 m/s.

The path following trajectory of the robot is shown in Figure 6.10, and the variations of the lateral and orientation errors are shown in Figures 6.11 and 6.12 respectively. The results show that the 1-m initial lateral error was quickly decreased to and stabilized within the 0.1-m range in 10 s. The orientation errors also reached a stable variation range of ±4° after the same time period. The steering angle input is shown in Figure 6.13. The curves indicate that the robot first steered to the maximum right, and then quickly adjusted the steering when the lateral errors were reduced to small values around zero. When the robot reached a stable state, the reference steering angles varied within a range of ±5° while the actual steering angles of the robot followed the reference commands with an error around 4° and a delay about 2 s.

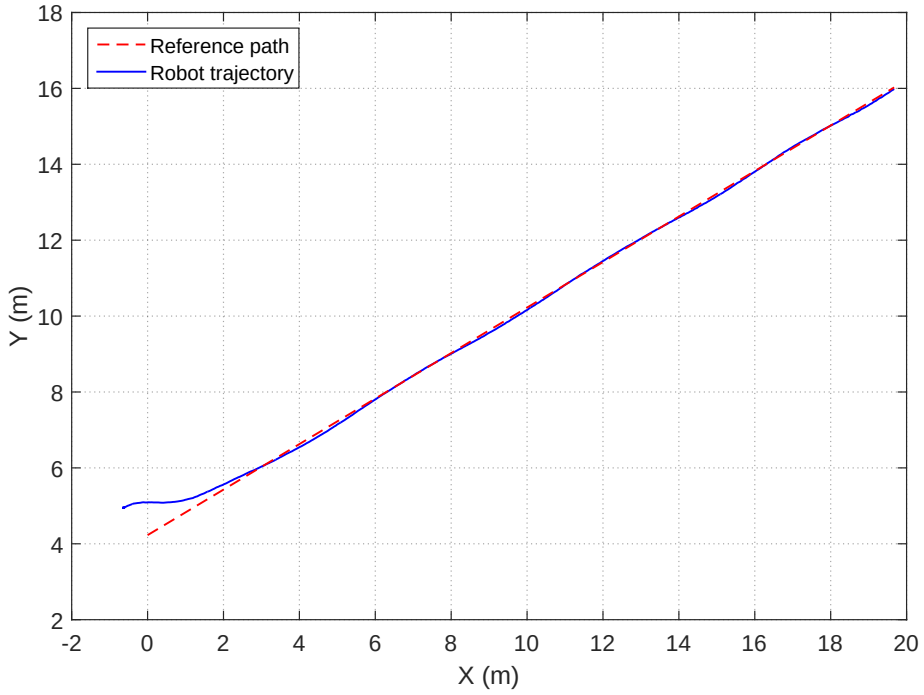


Figure 6.10: Path following trajectory, constant velocity with large initial deviation.

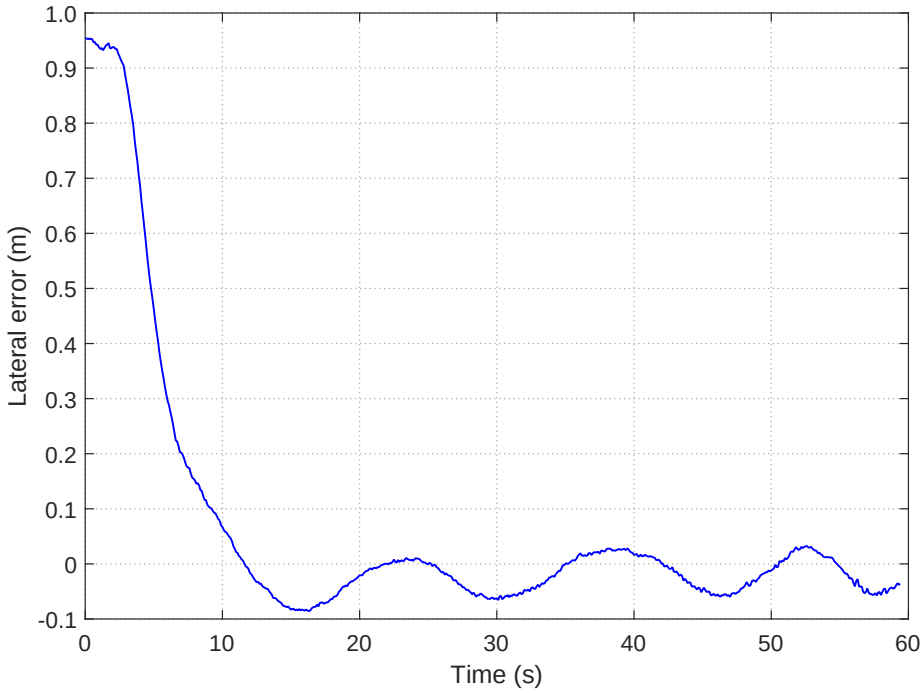


Figure 6.11: Lateral error, constant velocity with large initial deviation.

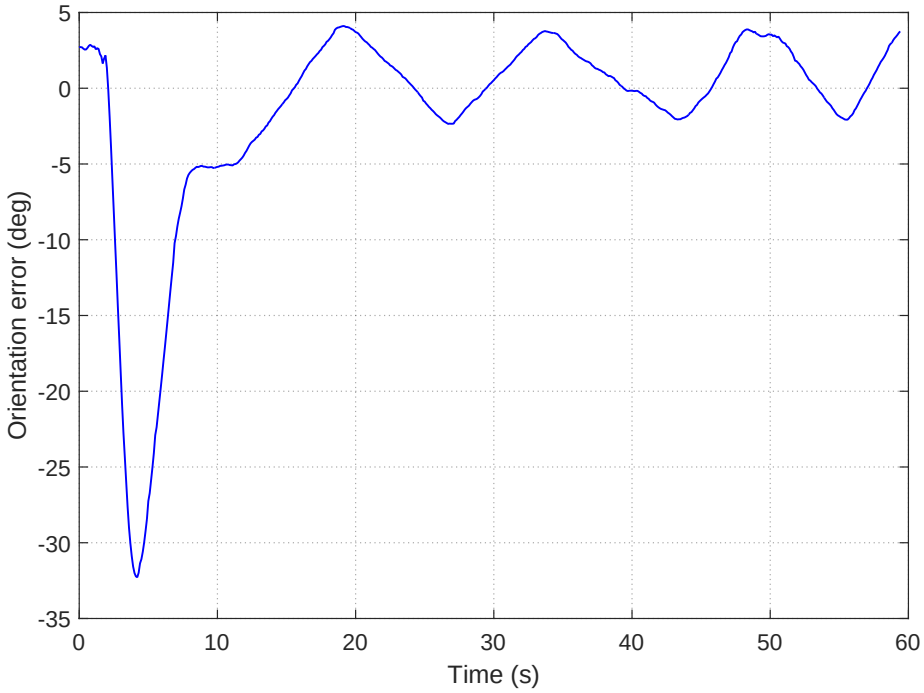


Figure 6.12: Orientation error, constant velocity with large initial deviation.

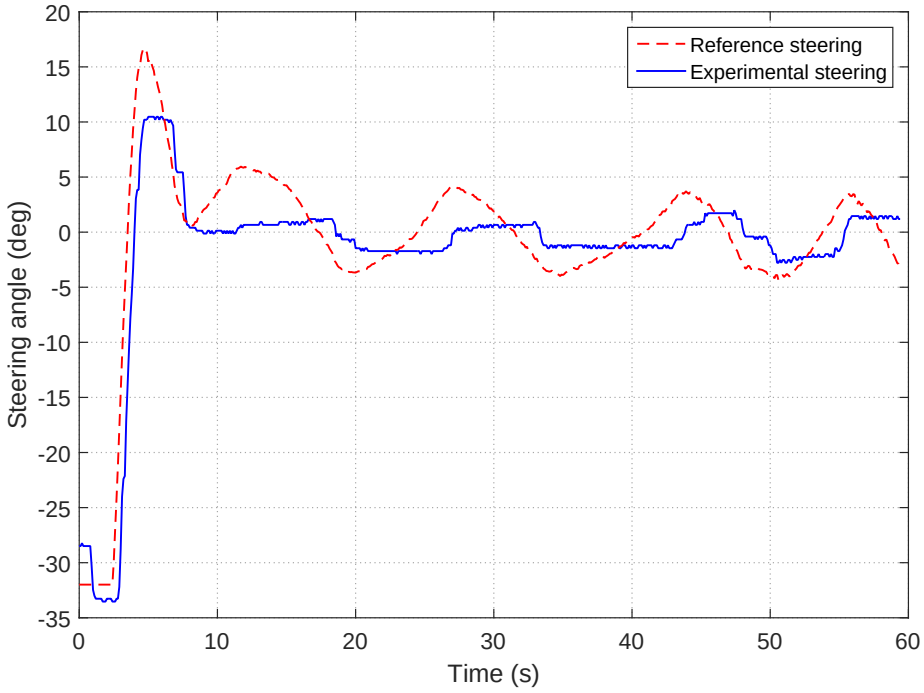


Figure 6.13: Steering angle input, constant velocity with large initial deviation.

6.1.2 Synchronous tracking

The second series of tests were performed to evaluate the synchronous tracking control. Two tests, one with a constant reference velocity and one with a varying reference velocity, were conducted. The control gains for the steering control law for both robots were kept as $k_1 = 2.4, k_2 = 1.4$, the same as in the single-robot path following experiments.

(1) Constant reference velocity

The reference velocity for the master robot was set to 0.3 m/s in this test. The control gain k_e for the slave velocity was set to 1.0. This value was determined by a few trial tests. A large k_e would result in large velocity variations for the slave robot and the offset error would fluctuate back and forth within a large range, whereas a too small k_e would make the slave unable to catch up with the master and a lag would always exist.

The trajectories of the synchronous tracking test are shown in Figure 6.14, and the lateral and orientation errors of the robots are shown in Figures 6.15 and 6.16 respectively. The steering angle inputs are shown in Figure 6.17.

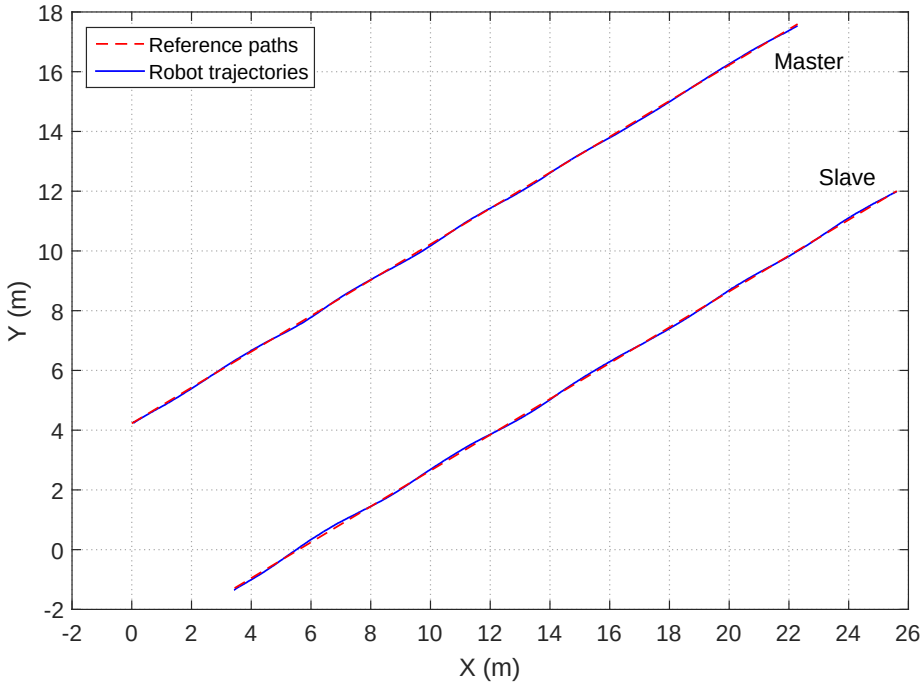
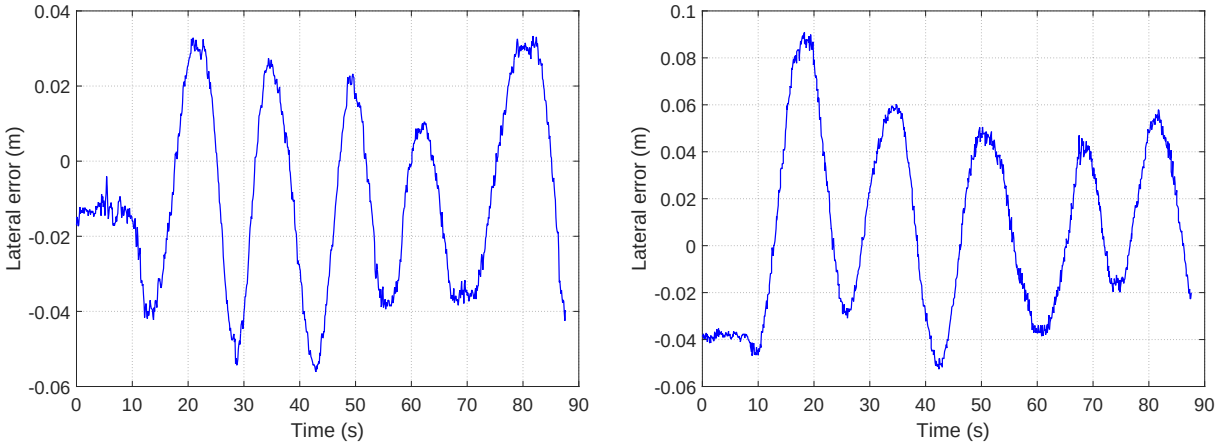


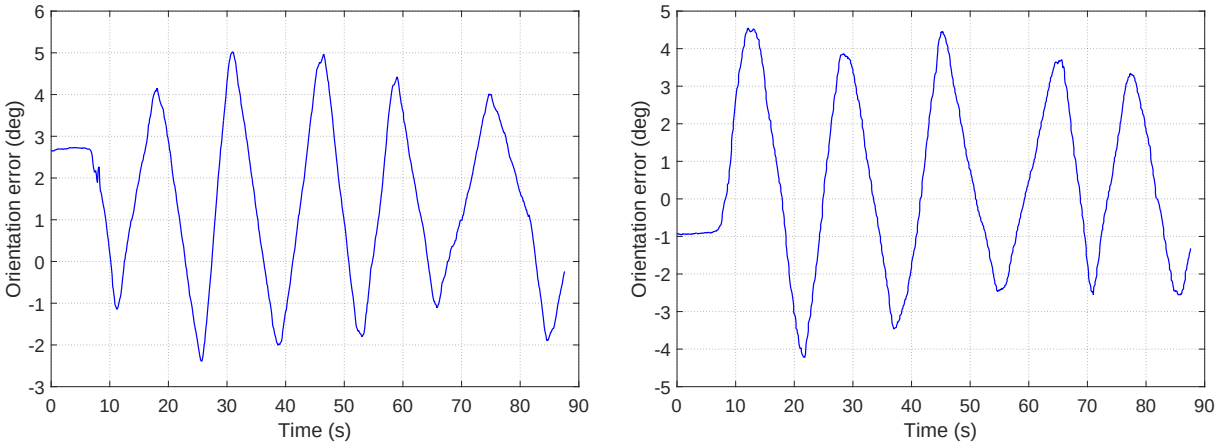
Figure 6.14: Synchronous tracking trajectories, constant reference velocity.

As the results show, both the master and the slave robots followed the reference paths with small lateral and orientation errors. The maximum lateral deviation was less than



(a) **(b)**

Figure 6.15: Lateral errors, constant reference velocity, (a) Master; (b) Slave.



(a) **(b)**

Figure 6.16: Orientation errors, constant reference velocity, (a) Master; (b) Slave.

0.06 m for the master and less than 0.1 m for the slave. Orientation errors were restricted in $\pm 5^{\circ}$ for both robots. These results indicate that the two robots performed as desired in their individual path following tasks.

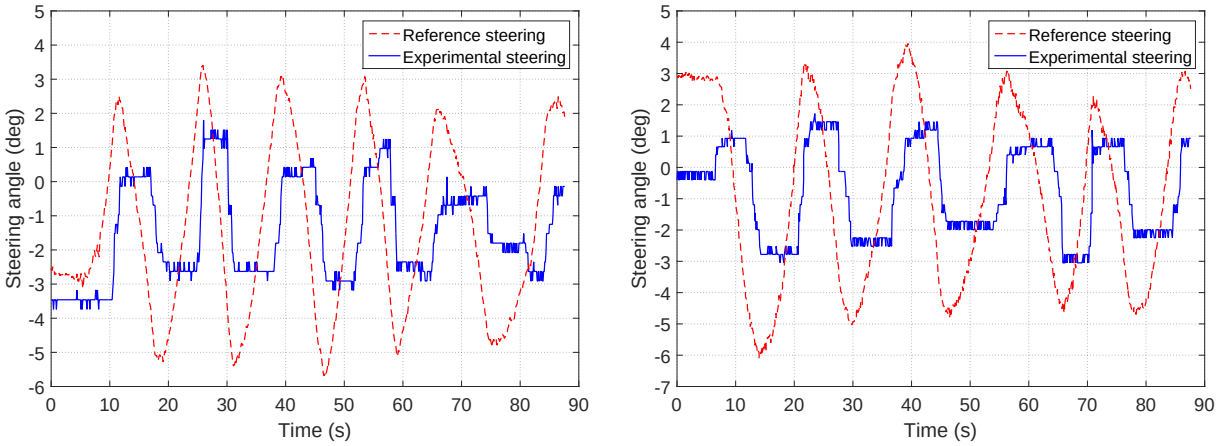


Figure 6.17: Steering angle inputs, constant reference velocity, (a) Master; (b) Slave.

The important requirement for the mobile mode is for the two vehicles to synchronize their motions during the operation. In the experiment, the velocity of the slave robot was controlled based on the real-time velocity of the master robot and the amount of their relative offset. The variation of the offset errors is plotted in Figure 6.18, and the velocity inputs for both robots are shown in Figure 6.19. The slave robot caught up with the master after some initial adjustments. The offset errors varied in a range of ± 0.05 m after both robots reached stable motions. It is noticeable that at the beginning stage a large offset error about 0.2 m was generated, and the slave velocity rose to 0.5 m/s to correct that error. This is because the master velocity was changed directly from 0 to 0.3 m/s when the control program started, and this sudden velocity change resulted in a large lead of the master. However, as Figure 6.19 shows, the slave robot quickly adjusted its velocity and matched it to the master's in a few seconds.

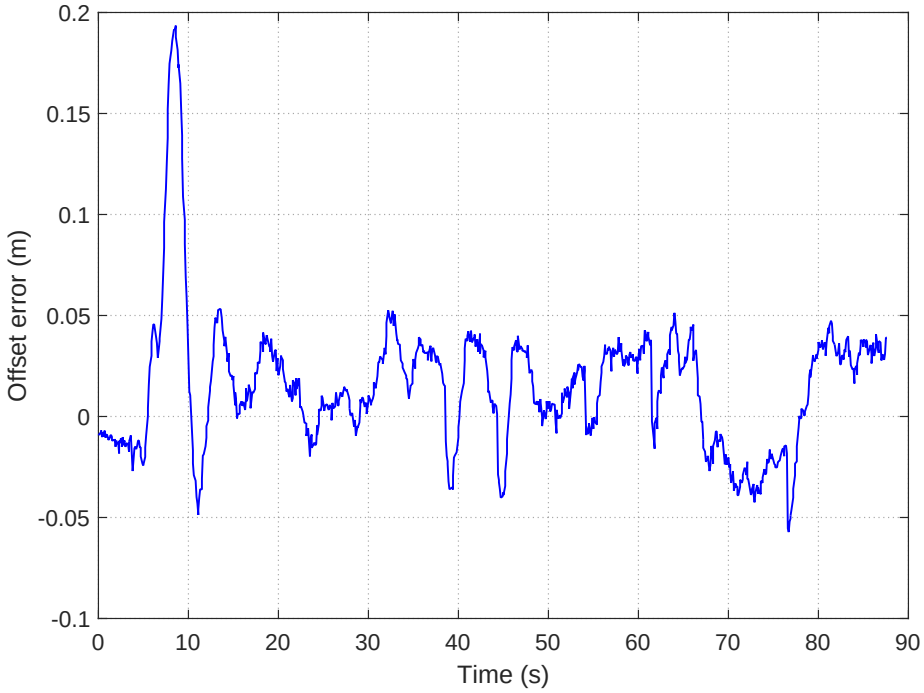


Figure 6.18: Offset error, constant reference velocity.

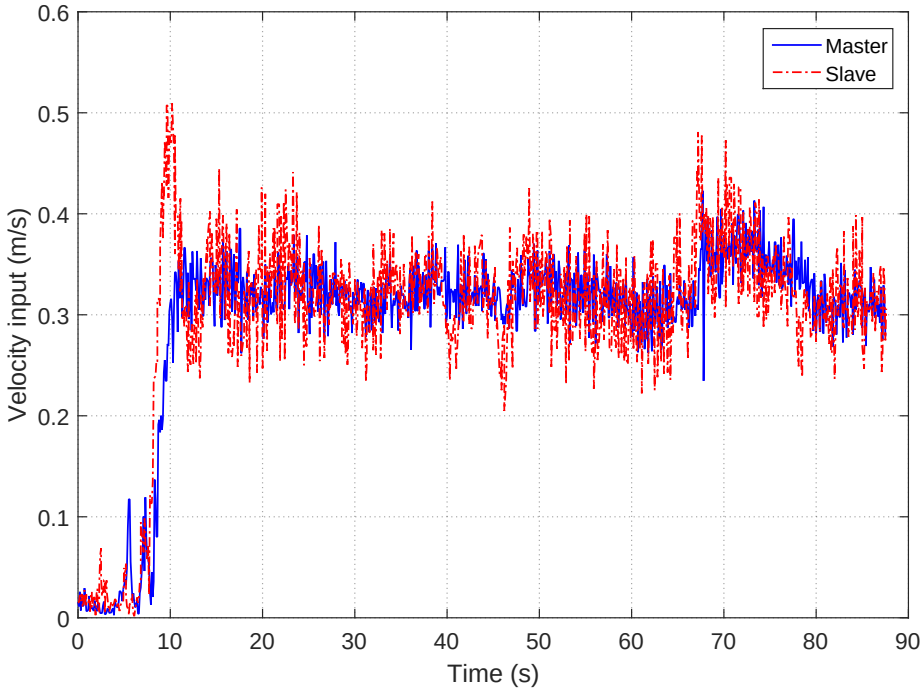


Figure 6.19: Constant velocity inputs.

(2) Varying reference velocity

Although agricultural operations are usually performed under constant velocities, to test the adaptability of the synchronous tracking control, the experiment with a varying reference velocity was performed. A sinusoidal operation velocity ranging from 0.2 to 0.6 m/s was set as the velocity input for the master robot. The control gain of the slave velocity was set to $k_e = 1.2$ after several trial tests.

The trajectories of the two robots in this experiment are shown in Figure 6.20. The lateral errors, orientation errors, and steering angle inputs are shown in Figures 6.21, 6.22, and 6.23, respectively. Similar to the constant reference velocity experiment, the lateral and orientation errors for both robots were kept less than 0.1 m and 5°, which indicates that desired path following performance was achieved for both robots.

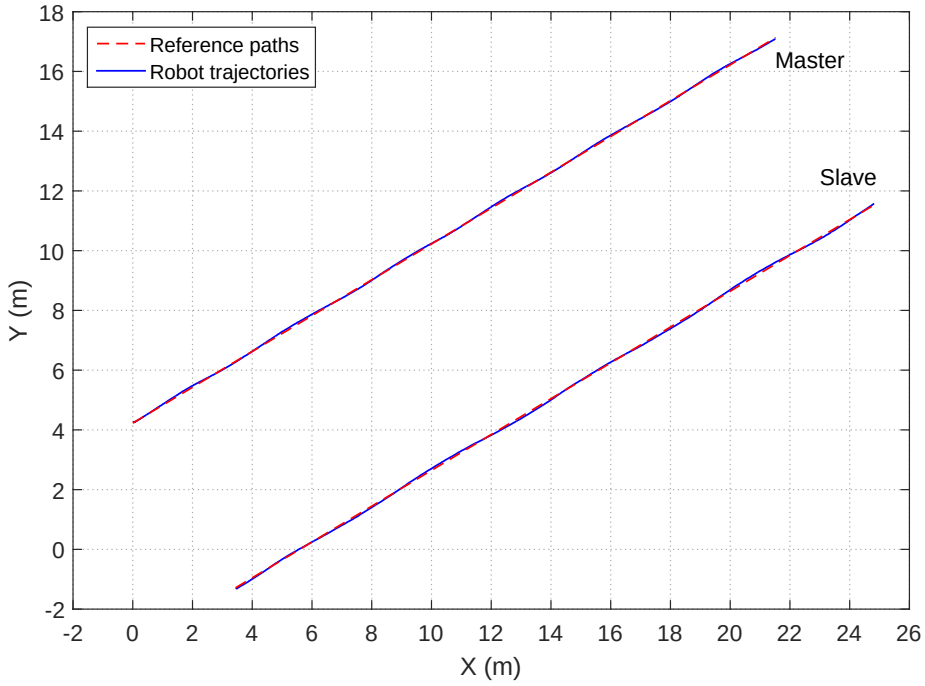
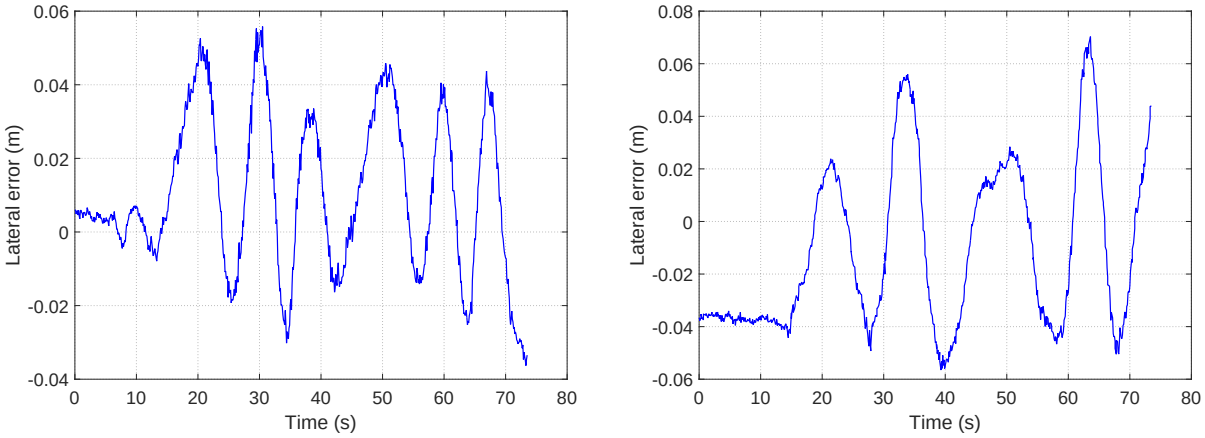
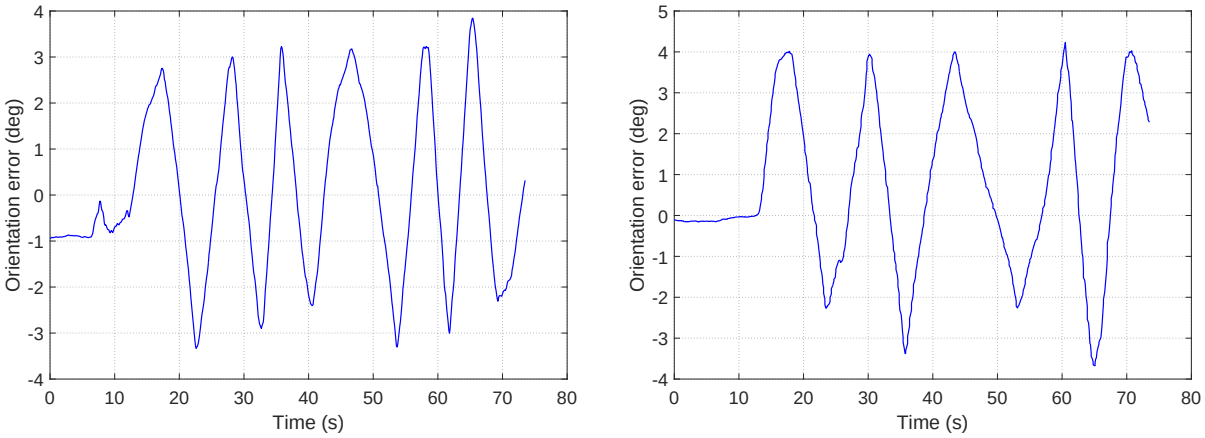


Figure 6.20: Synchronous tracking trajectories, varying reference velocity.



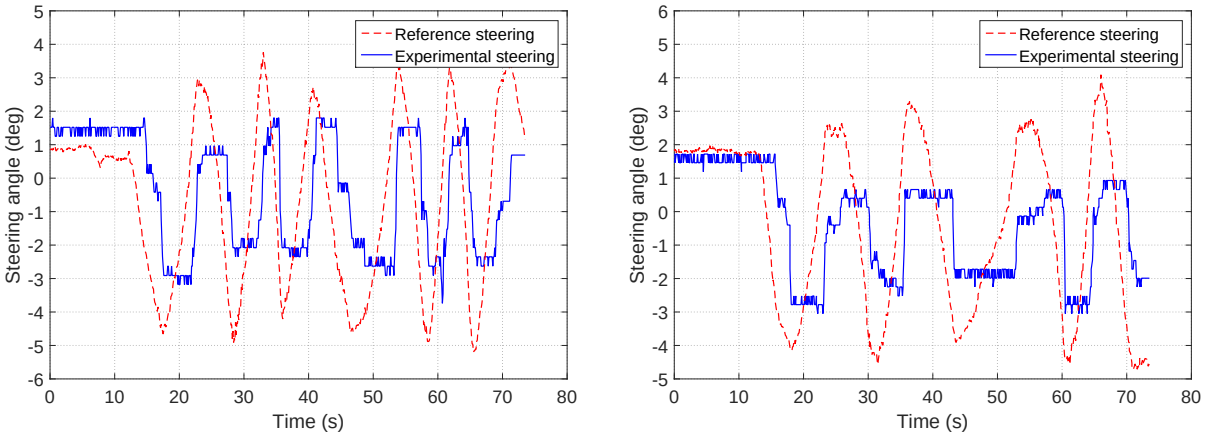
(a) **(b)**

Figure 6.21: Lateral errors, varying reference velocity, (a) Master; (b) Slave.



(a) **(b)**

Figure 6.22: Orientation errors, varying reference velocity, (a) Master; (b) Slave.



(a) **(b)**

Figure 6.23: Steering angle inputs, varying reference velocity, (a) Master; (b) Slave.

The offset error is shown in Figure 6.24. After the transient stage, the offset between the two robots was stabilized to the range (-0.05, 0.1) m. The velocity inputs of the two robots are shown in Figure 6.25. It can be seen that the slave velocity tracked the pattern of the master velocity well during the process, except for the initial velocity jump caused by the sudden start of the master. A larger lead was observed in this varying velocity experiment, compared with the constant velocity experiment. The maximum lead occurred when the master passed the average velocity 0.4 m/s and went on with an accelerating motion ($e = 0.08$ m at $t = 26.2$ s and $v_m = 0.46$ m/s, and $e = 0.09$ m at $t = 56.8$ s and $v_m = 0.49$ m/s).

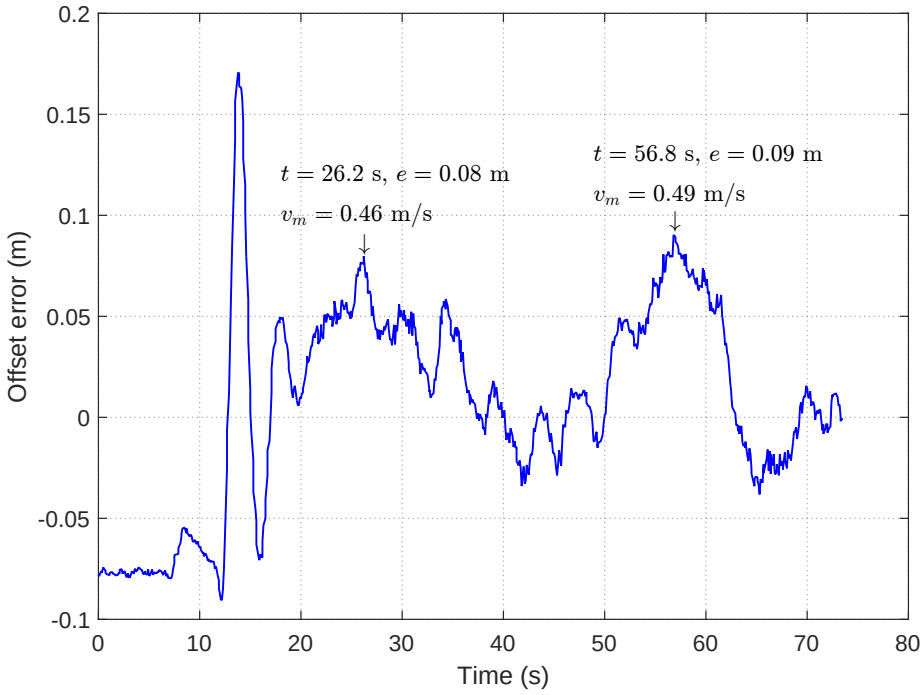


Figure 6.24: Offset error, varying reference velocity.

To evaluate the overall performance of the synchronous tracking control, statistical values of the errors were calculated. The lateral and orientation errors of the master and slave robots as well as the offset errors between the two are presented in Table 6.2. These values were calculated taking into account the stage of stable motions of both robots, and the initial adjustment stage was disregarded. For the calculations, a stable motion was defined as when the offset errors reached the stable variation range and were later kept in the range all time. It can be seen from the table that the path following errors of both robots were consistent with the results in Table 6.1. The average lateral error was around 0.01 m and the RMS lateral error was less than 0.04 m for both robots. During the

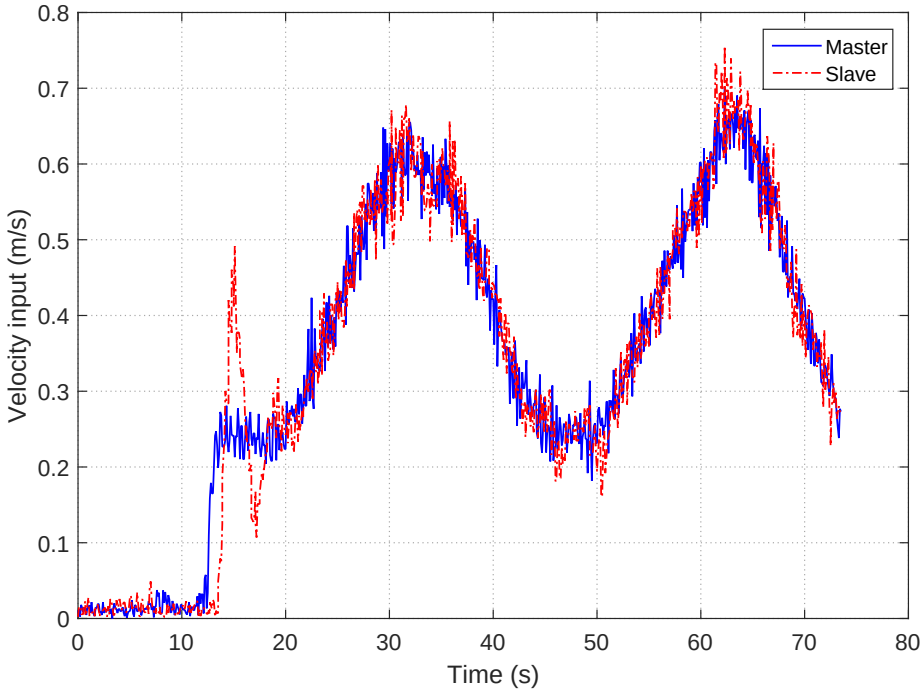


Figure 6.25: Varying velocity inputs.

Table 6.2: Synchronous tracking errors.

	constant v 0.3 m/s					varying v 0.2-0.6 m/s				
	d_m (m)	θ_{pm} (°)	d_s (m)	θ_{ps} (°)	e (m)	d_m (m)	θ_{pm} (°)	d_s (m)	θ_{ps} (°)	e (m)
Average	-0.01	1.33	0.01	0.55	0.01	0.01	0.17	0	0.79	0.02
RMS	0.03	2.39	0.04	2.43	0.03	0.03	1.97	0.03	2.32	0.04
Max	0.03	5.02	0.09	4.54	0.05	0.06	3.85	0.07	4.23	0.09
Min	-0.06	-2.39	-0.05	-4.22	-0.06	-0.04	-3.34	-0.06	-3.68	-0.05

Note: subscripts m and s stand for master and slave respectively.

whole tracking process, the lateral errors were kept less than 0.1 m. The average, RMS, and maximum orientation errors were smaller than 2°, 3°, and 5.5°, respectively. For the offset error e, the average, RMS, maximum lead, and maximum lag (listed in the table as Min) in the constant velocity experiment were 0.01 m, 0.03 m, 0.05 m, and 0.06 m, respectively. Slightly larger average, RMS, and maximum lead errors were observed in the varying velocity experiment, which were 0.02 m, 0.04 m, and 0.09 m, respectively. This offset error control performance is acceptable considering the velocity control accuracy for the robots, which was within 0.1 m/s, and the velocity measurement accuracy, which was 0.03 m/s.

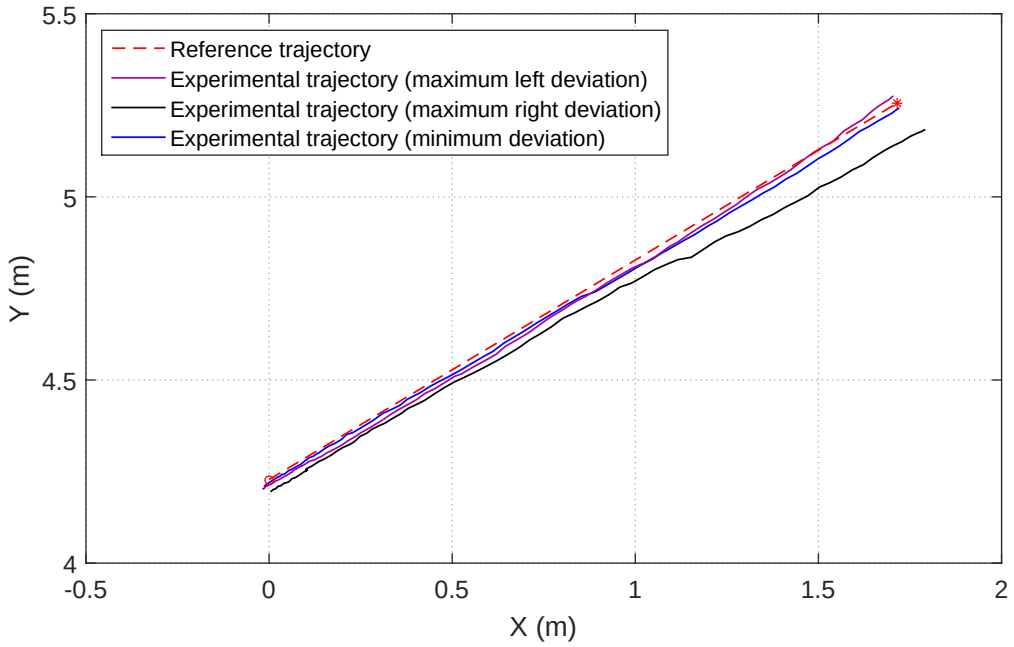
6.2 Stationary mode

6.2.1 Single point tracking

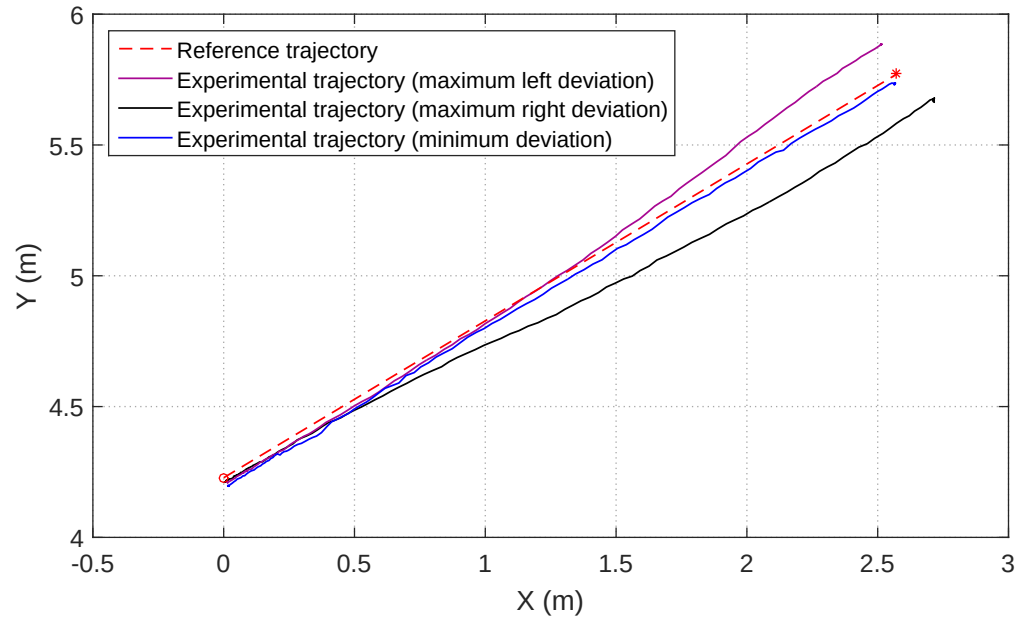
In the first series of tests, a single point tracking case was considered. The point with coordinates $(0, 4.228)$ was chosen as the start point. Two target points, $(1.715, 5.256)$ and $(2.573, 5.771)$, were determined on the path, which corresponded to a 2-m and a 3-m implement width respectively.

Before the experiments, a series of simulations were performed to determine the appropriate parameters k_v and n for the velocity equation and control gains k_1 and k_u for the steering angle equation. On the robot, the steering motor directly controls the front wheels through a rocker-arm linkage and some backlash exists between the connection. This makes the steering mechanism on the robot not as accurate and fast as the one on a modern tractor. The maximum speed that the robots should run at was restricted to 0.4 m/s. This value was determined by trial tests in which the robot steering was capable of correcting large lateral deviations. The order number $n = 8$ was used in the velocity equation so that the maximum acceleration was restricted to 0.3 m/s^2. The values of k_1 and k_u were determined as 0.5 and 0.6 respectively. To implement the modified control strategy, a virtual target point located 1 m further to the true one was used in the control system.

The tracking trajectories of the 2-m and 3-m implement width experiments are shown in Figure 6.26. The robots were placed near the desired start point with little initial errors. For each implement width, 15 experiments were repeated and the trajectories with maximum lateral deviations (left and right) and the minimum lateral deviation were drawn in the figure. The velocity and steering angle inputs for the trajectories with minimum lateral deviations are shown in Figures 6.27 and 6.28 respectively. Compared with the reference signals, the actual control inputs tracked the reference ones with errors that were within the calibrated error ranges. The steering angle inputs were small values around zero since the robots were placed near the desired start point with little initial errors.

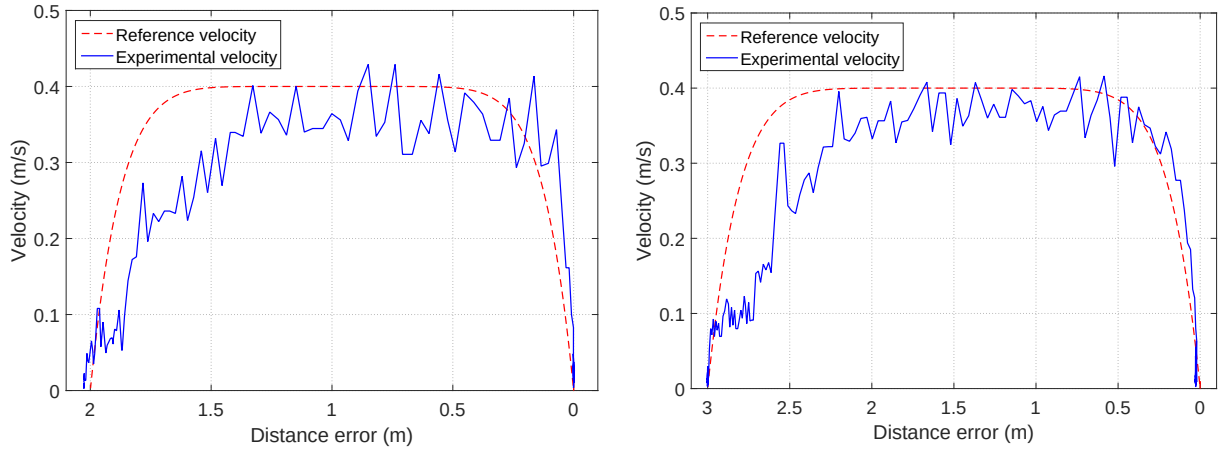


(a)



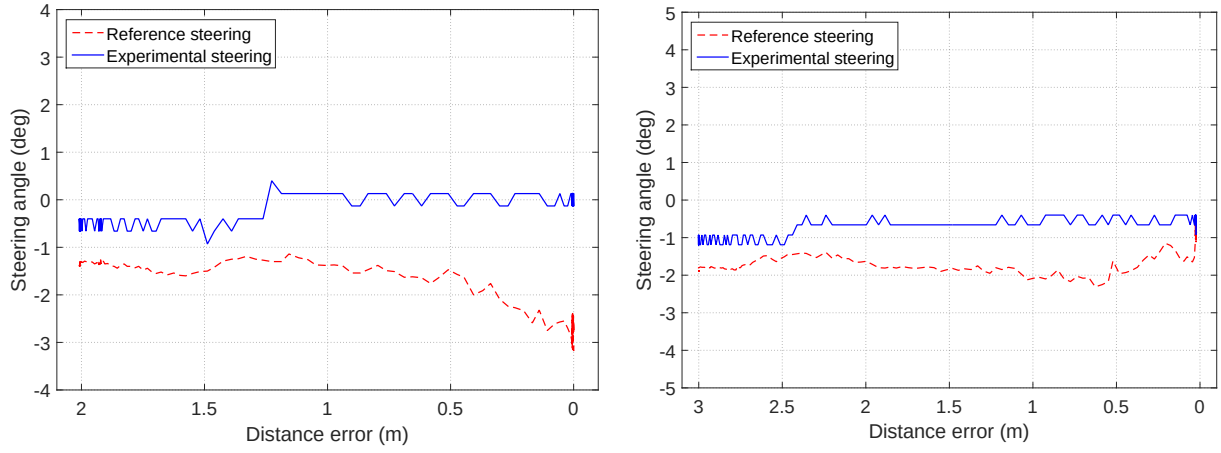
(b)

Figure 6.26: Tracking trajectories, (a) 2-m implement width; (b) 3-m implement width.



(a) **(b)**

Figure 6.27: Velocity inputs with respect to distance errors, (a) 2-m implement width; (b) 3-m implement width.



(a) **(b)**

Figure 6.28: Steering angle inputs with respect to distance errors, (a) 2-m implement width; (b) 3-m implement width.

To analyze the point tracking performance, the average, RMS, maximum, and minimum errors of the 15 runs were calculated and are presented in Table 6.3. The average offset error was -0.02 m for the 2-m experiments and 0 for the 3-m experiments. The RMS of the offset error for both groups of experiments was 0.04 m. The maximum offset errors were 0.02 m and 0.03 m, below the RMS, and the minimum were -0.10 m and -0.08 m. A negative offset error means overshoot of the forward distance. To prevent operation misses, a 0.1-m deduction can be applied in calculations of the target points. This means that given an implement width of w m, a value of $w-0.1$ m should be used as the base value for calculating target points. However, the accuracy of forward distances depends highly on the velocity control capability. For the experimental platform used in this study, the velocity control completely relied on the electric motor responses and no braking mechanism was applied when stopping. A more accurate closed-loop velocity control plus a braking mechanism on practical WSIC tractors could possibly improve the forward distance accuracy and a smaller deduction value could be used.

The average lateral errors for both experiments were less than 0.03 m and the RMS were less than 0.08 m. The lateral deviations were within a ± 0.16-m range, which is acceptable considering the path width, which is normally two times as wide as the vehicle tread width. The orientation errors reflect the posture deviations of the vehicle with respect to the reference path. As listed in the table, the maximum orientation error was less than 11° and both the average and the RMS were smaller than 4°.

Table 6.3: Point tracking errors.

	2-m implement width			3-m implement width		
	d (m)	e (m)	θ_e (°)	d (m)	e (m)	θ_e (°)
Average	-0.03	-0.02	2.34	0	0	2.75
RMS	0.05	0.04	3.66	0.08	0.04	3.94
Max	0.02	0.02	10.35	0.13	0.03	7.73
Min	-0.10	-0.10	-0.72	-0.16	-0.08	-2.50

In the above experiments, the robots were placed near the start point with little initial deviations and the steering control commands were small values around zero. To testify whether the designed steering control strategy can recover the vehicle from large initial errors, the experiment when the robot was placed with a 0.5-m initial lateral error was performed. Figure 6.29 shows the point tracking trajectory and Figure 6.30 shows the steering angle input. The figures indicate that the robot first steered to the right to correct the large initial lateral error. When the lateral error was reduced, the steering angle was gradually decreased to a value around zero. It is also noted that the actual steering angles lagged behind the reference signals.

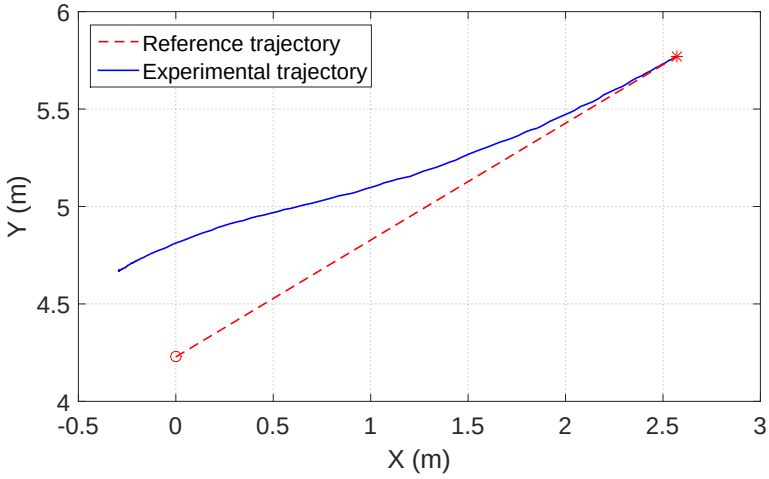


Figure 6.29: Tracking trajectory, 3-m implement width with 0.5-m lateral error.

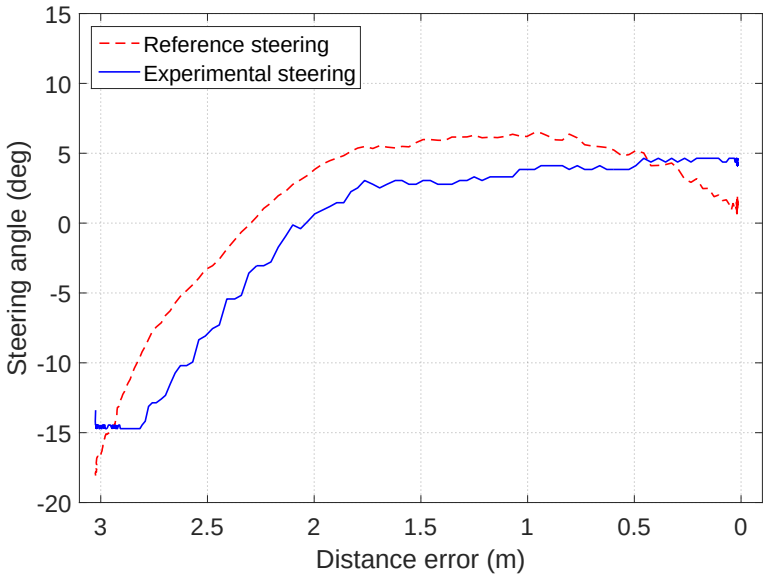


Figure 6.30: Steering angle input, 3-m implement width with 0.5-m lateral error.

6.2.2 Sequential point-to-point tracking

In the second series of tests, the designed control sequence was executed. The robot on path 1 (see Figure 5.14) was controlled as the master and the one on path 2 was controlled as the slave. A 2.5-m implement width was used and the coordinates of the start and target points were calculated. Table 6.4 lists the coordinates of the start points and four

groups of target points. The velocity input with $k_v = 0.3$ m/s and $n = 8$ was used and k_1 and k_u were set to 0.5 and 0.6 respectively.

Table 6.4: Coordinates of start and target points.

	Path 1	Path 2
Start point	(0, 4.228)	(3.343, -1.347)
Target #1	(2.144, 5.514)	(5.487, -0.061)
Target #2	(4.288, 6.799)	(7.631, 1.225)
Target #3	(6.432, 8.085)	(9.775, 2.510)
Target #4	(8.576, 9.371)	(11.919, 3.796)

In this sequential point-to-point tracking experiment, both robots needed to complete four point tracking processes. Three groups of tests were completed and the resulted tracking trajectories are shown in Figure 6.31. The detailed errors when the robots stopped at each target point are presented in Table 6.5. The average offset error was 0.02 m and the maximum lateral and orientation deviations were 0.16 m and 9.04°. These values are consistent with the error statistics summarized in Table 6.3. For experiment 1, the velocity and steering angle inputs with respect to the forward distances are shown in Figures 6.32 and 6.33 respectively.

Table 6.5: Sequential point-to-point tracking errors.

		Path 1/Master			Path 2/Slave		
		d (m)	e (m)	θ_e (°)	d (m)	e (m)	θ_e (°)
Experiment 1	Target #1	-0.04	0.02	1.94	0	0.02	1.56
	Target #2	-0.08	0.02	3.35	0.16	0.02	-9.04
	Target #3	0	0.02	2.54	0.07	0.01	-2.35
	Target #4	-0.11	0.02	3.83	0.05	0.02	-5.33
Experiment 2	Target #1	0	0.02	-2.05	-0.04	0.02	2.03
	Target #2	-0.15	0.02	8.36	0.12	0.02	-4.84
	Target #3	-0.03	0.02	4.88	-0.02	0.02	-1.31
	Target #4	-0.14	0.02	5.39	0.10	0.02	-3.11
Experiment 3	Target #1	0.05	0.02	-0.15	0.06	0.02	1.78
	Target #2	-0.03	0.02	-0.85	0	0.02	-2.55
	Target #3	0.11	0.02	-3.37	0.09	0.01	-0.82
	Target #4	-0.02	0.02	-1.05	0	0.02	-3.03

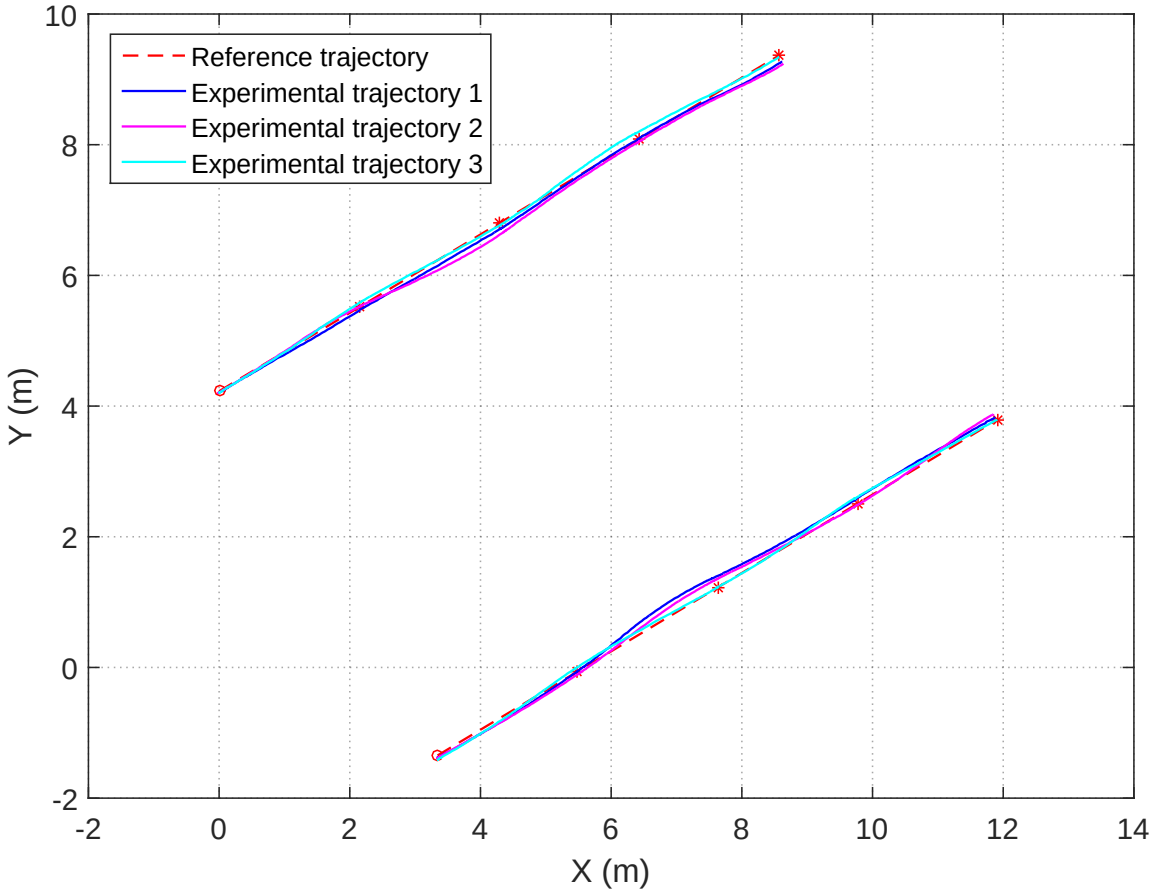
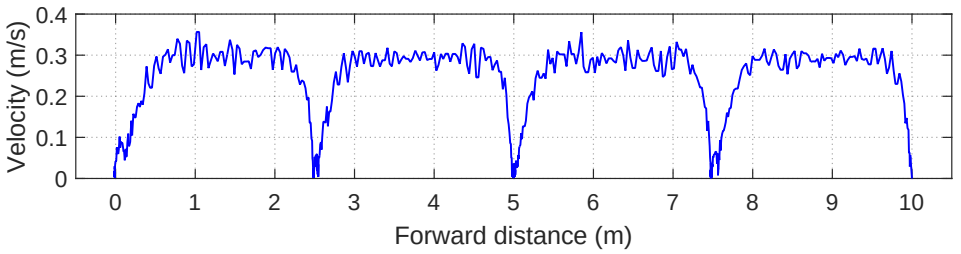
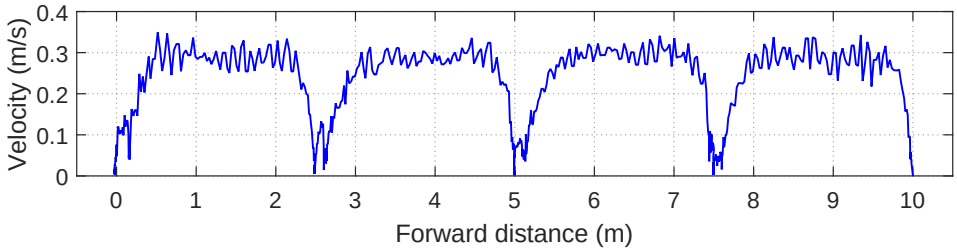


Figure 6.31: Sequential point-to-point tracking trajectories.

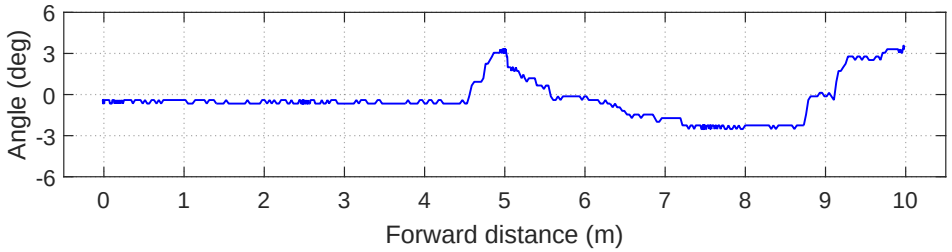


(a)

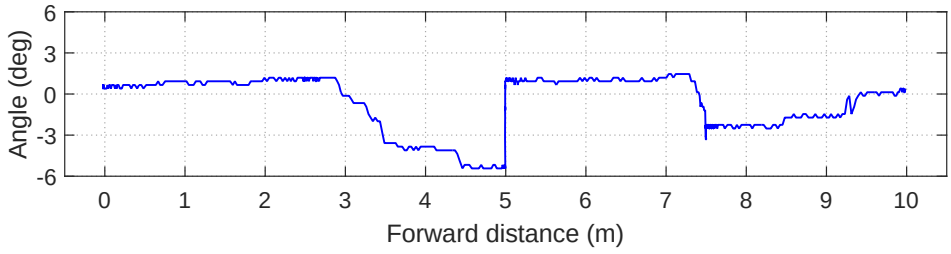


(b)

Figure 6.32: Velocity inputs with respect to forward distances, (a) Master; (b) Slave.



(a)



(b)

Figure 6.33: Steering angle inputs with respect to forward distances, (a) Master; (b) Slave.

Figure 6.34 shows the time sequence of the forward distances for the master and slave robots in experiment 1. The timeline (in the form of HH:MM:SS.X, 0.1 s accuracy) was extracted from the saved data files. When running the control programs, the onboard laptop saved the control signals, errors, and the current time strings. The implement operation was simulated by a constant pause of 12 s in the program. In the control scheme, the master initiates the start of tracking processes for both vehicles, and the implement can only start operating when both vehicles reach a full stop. It is desirable to have both vehicles finish the tracking processes at the same time to ensure efficient operation. However, a variety of factors influence the length of time that the robots take to perform the point tracking processes; some important ones include the driving motor responsiveness, the flatness and friction of the surface, and the velocity control accuracy and measurement accuracy. In Figure 6.34, the time differences when the master and slave stopped at targets 1, 2, 3, and 4 were 0.6 s, 1.2 s, 0.9 s, and 0.6 s, respectively. The time sequence shows that the motions of the master and the slave were in good synchronization.

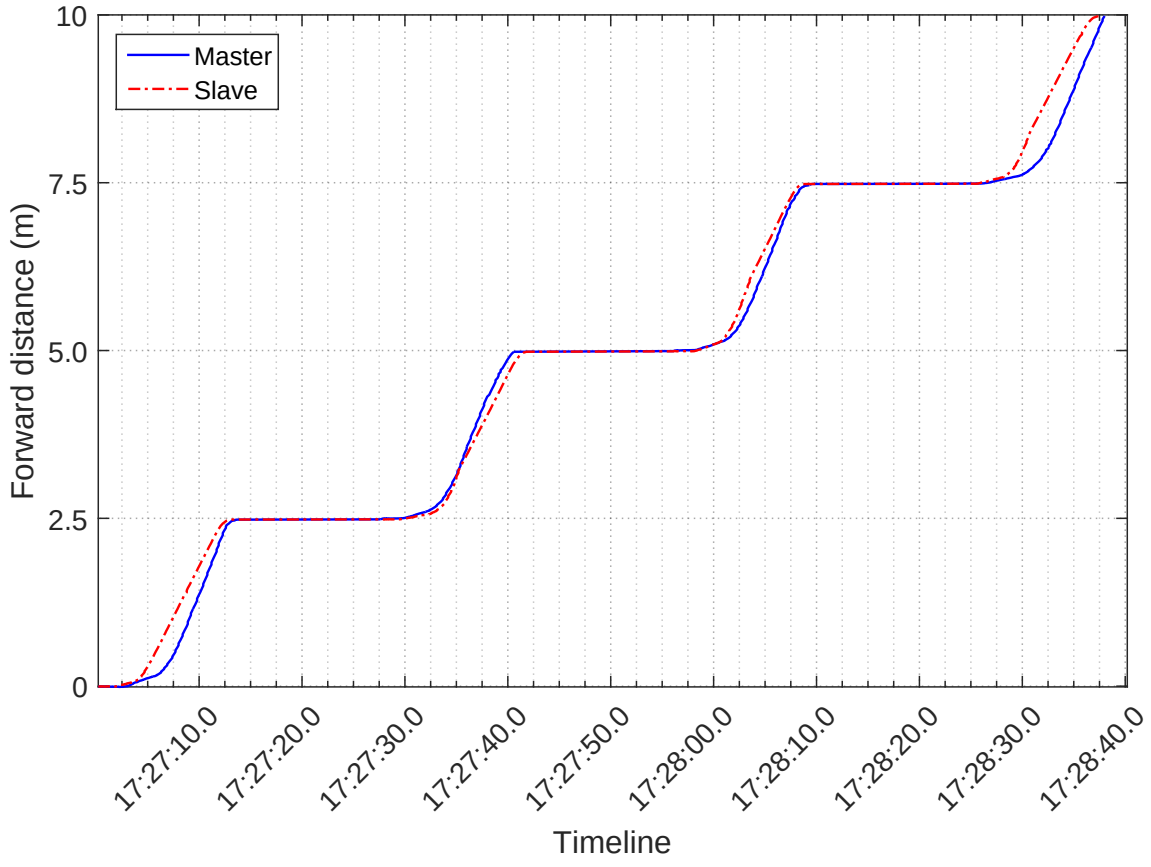


Figure 6.34: Time sequence of forward distances.

Chapter 7

Conclusions

7.1 Summary of contributions

Automation of agricultural platforms is playing an increasingly important role in modern agricultural production. A number of automatic guidance and control systems have been developed and applied to cope with dwindling labour force in agriculture and increasing demands for precision farming. The WSIC, which applies the CTF concept, is a promising platform to solve problems, such as damages to crops and excessive soil compaction, that can be induced by traditional tractor-based operation systems. Its application in cranberry production over the past two decades has demonstrated its versatility and productivity. However, a main limitation of the WSIC platform is its high demands for labour force input, two to three operators required at one time. In addition, the pure manual guidance and control cannot guarantee its operation accuracy and efficiency.

The fundamental new achievement of this research is the first ever design, implementation, and evaluation of an automatic guidance system for the WSIC. Through the development of control models and algorithms for both the mobile and the stationary operating modes, the development of a robotic experimental platform, and experimental validations of the guidance and control methods, an automatic guidance system with decent accuracy and efficiency has been achieved. The main contributions of this thesis can be concluded as follows.

(1) Design of the control algorithms for the mobile operating mode of the WSIC.

The mobile operating mode of the WSIC was modelled as a synchronous trajectory tracking problem. Under this mode, the two WSIC tractors need to follow two predefined reference paths individually while at the same time keeping their motions synchronized. A master-slave cooperative method was designed for this problem. To make the master and

slave vehicles follow their reference paths, a path following task was modelled based on the classic kinematic bicycle model, and the control law for the front-wheel steering angle was designed to eliminate the lateral errors during the following process. Since the designed steering control law was velocity independent, the velocity of the slave vehicle could be controlled to synchronize its motion with the master's. An additional synchronous tracking model was developed for the slave, and a velocity control law was designed. This cooperative control method could guide the vehicles on their individual paths and meanwhile synchronize their forward motions.

The designed control algorithms were tested through simulations. A MATLAB/Simulink model was developed for the master-slave cooperative control. The simulations were performed under three scenarios, the first with a constant reference velocity, the second with a varying reference velocity, and the third with curved reference paths. Initial lateral and orientation errors of the vehicles to the reference paths and the initial offset error between the two vehicles were assumed in the simulations. For all the scenarios, the master and slave vehicles could eliminate their initial path following errors under the designed steering control, and the offset error was also diminished to zero when the slave vehicle matched its forward motion with the master's under the designed velocity control.

(2) Design of the control algorithms for the stationary operating mode of the WSIC.

The stationary operation mode of the WSIC was modelled as a parallel point-to-point tracking problem. Under this mode, both tractors need to perform a forward point tracking task and repeat the task in every operation cycle. A second master-slave cooperative method was designed for this problem, and two control tasks were defined. The first task was the single point tracking process and the second task was the periodic operating process. A point tracking model was developed based on the polar coordinates representation, and control laws for the forward velocity and the steering angle were designed. The velocity control law took the distance error from the start point to the target point as the control variable, and its profiles were in general form of parabola curves. This velocity control law allowed the vehicle to track the target points with only forward motions to ensure high operation efficiency. The steering control law was designed based on the backstepping control method to eliminate the heading and orientation errors of the point tracking task. Since the heading and orientation error angles were defined based on the existence of a distance error, directly applying the steering control law on a practical platform would result in irrational steering commands when the vehicle reaches the target point. To avoid this behaviour, modified control strategies were proposed by extending the true target point to a virtual one, which could make the heading and orientation errors always well defined and the steering inputs always reasonable.

A control sequence was designed for the periodic operating process and a master-slave cooperative method was applied. During the operation, the master vehicle controlled the start of forward point tracking processes and the start of implement operations. The slave vehicle performed point tracking processes under the control signals sent from the master. The two reference paths were divided into small sections according to the working width of the implement. A series of target points were then generated with their coordinates calculated and stored in the control system. An index number was added to each pair of target points as the control variable in the sequence execution.

The designed control laws for the point tracking task and the control sequence for the periodic operating process were tested through simulations. A MATLAB/Simulink model was developed for the forward point tracking process. Single point tracking cases with different initial conditions were simulated and the theoretical effectiveness of the velocity and steering angle control laws was verified. The point tracking model was then applied in the control sequence to complete five consecutive point tracking processes. Results show that both the master and the slave vehicles could eliminate their initial tracking errors in the first tracking process, and their motions were in perfect synchronization under ideal simulation conditions.

(3) Design of the automatic guidance system adapted to WSIC operations and of the associated hardware and software.

To test the designed control algorithms in practice, an experimental WSIC platform and the corresponding hardware and software of the automatic guidance system were developed. Although the experimental platform was a reduced-size one, the guidance system was developed from the consideration of a practical WSIC. A dual-rover RTK-GPS was used for the real-time positioning of the two WSIC vehicles, one rover on each vehicle. The positions of the vehicles could be conveniently gathered and compared to the reference paths and to each other. Two 3DM-GX3-25 IMUs were used to measure the real-time orientations of the vehicles during the automatic guidance process. Data communication between the two vehicles was required to implement the master-slave cooperative control. This was established wirelessly using two XBee-PRO modules considering the wide distance between the two WSIC vehicles. A series of control processors were used for the experimental platform. On each robot, a laptop PC, an Arduino microcontroller, and a Sabertooth motor driver were installed. The laptop PC functioned as the central control unit to communicate with the GPS, IMU, XBee, and Arduino and run control algorithms. The software of the guidance system was developed in the MATLAB environment in the host laptop, and a series of programs were written for interfacing with the peripheral hardware. The control structure of the guidance system was developed at two levels. The upper level was implemented on the laptop for calculating guidance algorithms and outputting velocity

and steering angle values. The lower level was executed on the Arduino microcontroller and the motor driver for receiving desired velocity and steering angle commands and controlling the motors.

(4) Experimental validation of the control algorithms and of the performance of the automatic guidance system.

To verify the effectiveness of the designed control algorithms, validation tests were performed using the experimental platform on the top floor of an outdoor parking garage. The experiments for the mobile mode were planned in two steps. In the first step, the path following control for a single robot was tested and proper control gains for the steering angle control law were suggested. The results show that the robot could follow the reference path with less than 0.1 m lateral error and less than 5° orientation error. The experiments in the second step tested the master-slave cooperative control for the synchronous tracking of the two robots. Two experiments were performed, the first with a constant reference velocity of 0.3 m/s and the second with a varying reference velocity ranging from 0.2 to 0.6 m/s. In both experiments, the path following performance of the two robots was consistent with the single-robot tests completed in the first step. The offset errors between the robots were restricted within 0.05 m in the constant-velocity experiment and within 0.1 m in the varying-velocity experiment.

The experiments for the stationary mode were planned in two steps as well. In the first step, a single point-to-point tracking case was considered and experiments of two implement widths, 2 m and 3 m, were performed. From the results, both the average and the RMS of the offset errors were less than 0.04 m, and the maximum lateral error was within 0.16 m and the maximum orientation error was within 11°. The master-slave cooperative control sequence was tested in the second step. Both the master and the slave robots were required to complete four point tracking processes with an implement width of 2.5 m. Test results show that the master and slave robots could complete the point tracking processes in a well-synchronized manner. The time differences when they stopped at their target points were less than 2 s. This motion synchronization could permit high efficiency during autonomous WSIC operations.

The lateral deviations were within 0.1 m in the mobile mode experiments and were within 0.16 m in the stationary mode experiments. This level of guidance accuracy is acceptable considering the width of the reference path on a cranberry farm, which is typically equivalent to twice the tread width of the WSIC tractor. The motion synchronization requirement is more crucial in the mobile mode than in the stationary mode. A maximum offset error of 0.09 m was observed in the experiments. This level of control accuracy is deemed acceptable and reasonable considering the velocity control accuracy of the robots

(0.1 m/s) and the velocity measurement accuracy of the GPS receivers (0.03 m/s).

7.2 Limitations

The performance of the developed automatic guidance system has been tested in validation experiments. However, since the tests were performed under some ideal conditions and on a scaled-down robotic platform, certain limitations exist in the current research.

(1) Control designs for the WSIC were based on ideal kinematic models. This simplification ignored the complex dynamic effects of the WSIC tractors and the truss, and allowed the validations to be performed on a scaled-down robotic platform without a connecting truss. The proposed control methods are applicable only when WSIC operations are performed under relatively ideal conditions including flat and firm travel paths and low operating speeds.

(2) The mobile robots used for the experimental validations were capable of rear-wheel driving and front-wheel steering. They were constructed to model the control behaviours of WSIC tractors. However, the steering mechanism on the robots was not as accurate and fast as the steering system on a modern tractor. This limited the maximum velocity that could be applied in the validation experiments. In addition, compared with large-size tractors, the small wheelbase of the robots made them more sensitive to steering errors and delays.

7.3 Future studies

The ultimate goal of developing an automatic guidance system for the agricultural WSIC is to replace the manual guidance and control process to improve its operation accuracy and efficiency. Although the guidance system described in this thesis was tested to be effective under some ideal conditions, improvements have to be made before the designed system can be applied on a practical field-scale WSIC. Future work can be performed in the following directions.

(1) An automatic guidance system has to be safe and reliable under all circumstances. The safety and emergency considerations are especially important for large-size agricultural platforms such as WSICs. It is crucial for the WSIC to have the ability to cope with emergencies such as loss of GPS signals, malfunction of steering control or velocity control, and loss of wireless communication between the master and the slave vehicles. These situations were not considered in the current development since the small-scale robotic

platform was easy to control and the running speeds were kept small. An emergency system, in forms of both hardware and software, has to be designed and tested for the practical WSIC and combined into the current guidance system.

(2) The robotic experimental platform has a few limitations as described above. To further test the performance of the guidance system, experiments can be performed on full-scale tractors with superior steering mechanisms. In the validation experiments, the control gains for the steering and velocity control laws were tuned and determined through a few trial tests. This way of choosing control gains is relatively inefficient for practical applications. Since agricultural operations cover a range of velocities, it is necessary to perform an integral study of determining proper control gains for different operation conditions. A control system capable of varying control gains to different operation velocities and initial conditions could be designed to improve the adaptability of the guidance system.

(3) Kinematic vehicle models are suitable only when ideal conditions including flat and firm travel paths and slow travel speeds are met or can be approximated. Under such conditions, the dynamics of the platform body and the slipping and skidding of the tires can be neglected. Since these ideal conditions are not perfectly met in practice, it is necessary to include some important dynamic effects in the guidance models to permit high accuracy of the guidance system. Although a full dynamic description of the WSIC would be too complex, a simplified analysis for the connections between the truss and the tractors and the contact between the tires and the ground can be performed. The rotating and sliding joints connecting the truss and the tractors will have to withstand shearing stresses from both horizontal and vertical directions when the tractors are travelling on uneven paths and their motions are not well synchronized. It is worthwhile to perform a force-and-stress analysis for the connections so that the influence of these forces and stresses can be determined. To maintain a high guidance accuracy when the WSIC works under non-ideal conditions, the slipping and skidding of the tires should be incorporated into the control model. A mathematical model for the contact between the tractor and the surface can be developed. Parameters such as the tire cornering stiffness, surface levelness, surface firmness and friction can be modelled to describe the slipping and skidding effects. Simulation studies can first be conducted using software such as Adams and MATLAB/Simulink. Different operation and path conditions can be modelled in the simulations, and suggestions can be given to the safety of autonomous WSIC operations.

References

Ahamed, T., Takigawa, T., Koike, M., Honma, T., Yoda, A., Hasegawa, H., ... Zhang, Q. (2004). Characterization of laser range finder for in-field navigation of autonomous tractor. In *Proceedings of the Conference on Automation Technology for Off-road Equipment. Kyoto, Janpan.*

Ahamed, T., Tian, L., Takigawa, T., & Zhang, Y. (2009). Development of auto-hitching navigation system for farm implements using laser range finder. *Transactions of the ASABE*, *52*(5), 1793-1803.

Aicardi, M., Casalino, G., Balestrino, A., & Bicchi, A. (1994). Closed loop smooth steering of unicycle-like vehicles. In *Proceedings of the 33rd IEEE Conference on Decision and Control* (Vol. 3, p. 2455-2458).

Aicardi, M., Casalino, G., Bicchi, A., & Balestrino, A. (1995). Closed loop steering of unicycle-like vehicles via Lyapunov techniques. *IEEE Robotics & Automation Magazine*, *2*, 27-35.

Astolfi, A. (1999). Exponential stabilization of a wheeled mobile robot via discontinuous control. *Journal of Dynamic Systems, Measurement, and Control*, *121*(1), 121-126.

Bayar, G., Bergerman, M., Koku, A. B., & ilhan Konukseven, E. (2015). Localization and control of an autonomous orchard vehicle. *Computers and Electronics in Agriculture*, *115*, 118-128.

Bell, T. (2000). Automatic tractor guidance using carrier-phase differential GPS. *Computers and Electronics in Agriculture*, *25*(1-2), 53-66.

Bell, T., O'Connor, M., Jones, V. K., Rekow, A., Elkaim, G., & Parkinson, B. (1998). Realistic autofarming closed-loop tractor control over irregular paths using kinematic GPS. *Journal of Navigation*, *51*(3), 327-335.

Benson, E. R., Reid, J. F., & Zhang, Q. (2003). Machine vision-based guidance system for an agricultural small-grain harvester. *Transactions of the ASAE*, *46*(4), 1255-1264.

Brockett, R. W. (1983). Asymptotic stability and feedback stabilization. *Differential Geometry Control Theory*, 181-191.

Cariou, C., Lenain, R., Thuilot, B., & Berducat, M. (2009). Automatic guidance of a four-wheel-steering mobile robot for accurate field operations. *Journal of Field Robotics*, *26*(6-7), 504-518.

Chamen, W., Watts, C., Leede, P., & Longstaff, D. (1992). Assessment of a wide span vehicle (gantry), and soil and cereal crop responses to its use in a zero traffic regime. *Soil & Tillage Research*, *24*, 359-380.

Chateau, T., Debain, C., Collange, F., Trassoudaine, L., & Alizon, J. (2000). Automatic guidance of agricultural vehicles using a laser sensor. *Computers and Electronics in Agriculture*, *28*(3), 243-257.

Chen, B., Tojo, S., & Watanabe, K. (2003). Machine vision based guidance system for automatic rice transplanters. *Applied Engineering in Agriculture*, *19*(1), 91-97.

Chen, C.-Y., Li, T.-H. S., Yeh, Y.-C., & Chang, C.-C. (2009). Adaptive neuro-fuzzy network control for a mobile robot. *Mechatronics*, *19*(2), 156-166.

Chwa, D. (2004). Sliding-mode tracking control of nonholonomic wheeled mobile robots in polar coordinates. *IEEE Transactions on Control Systems Technology*, *12*(4), 637-644.

Cole, A. (2008). Integrating external position information with GPS using existing GPS processing software for precision agriculture applications. In *Proceedings of ION GNSS, Savannah, Georgia* (p. 2156-2164).

Corradini, M. L., & Orlando, G. (2001). Robust tracking control of mobile robots in the presence of uncertainties in the dynamical model. *Journal of Robotic Systems*, *18*(6), 317-323.

D'Andréa-Novel, B., Campion, G., & Bastin, G. (1995). Control of nonholonomic wheeled mobile robots by state feedback linearization. *The International Journal of Robotics Research*, *14*(6), 543-559.

De Luca, A., Oriolo, G., & Samson, C. (1998). Feedback control of a nonholonomic car-like robot. In J.-P. Laumond (Ed.), *Robot motion planning and control* (Vol. 229, p. 171-253). Springer Berlin Heidelberg.

De Wit, C., & Sørdalen, O. (1992). Exponential stabilization of mobile robots with nonholonomic constraints. *IEEE Transactions on Automatic Control*, *37*(11), 1791-1797.

Fang, H., Dou, L., Chen, J., Lenain, R., Thuilot, B., & Martinet, P. (2011). Robust anti-sliding control of autonomous vehicles in presence of lateral disturbances. *Control Engineering Practice*, *19*(5), 468-478.

Fang, H., Fan, R., Thuilot, B., & Martinet, P. (2006). Trajectory tracking control of farm vehicles in presence of sliding. *Robotics and Autonomous Systems*, *54*(10), 828-839.

Fang, H., Lenain, R., Thuilot, B., & Martinet, P. (2005). Robust adaptive control of automatic guidance of farm vehicles in the presence of sliding. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005* (p. 3102-3107).

Feng, L., He, Y., Bao, Y., & Fang, H. (2005). Development of trajectory model for a tractor-implement system for automated navigation applications. In *Proceedings of IEEE Instrumentation and Measurement Technology Conference, Ottawa, Canada* (p. 1330-1334).

Fierro, R., & Lewis, F. L. (1995). Control of a nonholomic mobile robot: Backstepping kinematics into dynamics. In *Proceedings of the 34th Conference on Decision and Control. New Orleans, LA* (p. 3805-3810).

Fierro, R., & Lewis, F. L. (1997a). Control of a nonholomic mobile robot: Backstepping kinematics into dynamics. *Journal of Robotic Systems*, *14*(3), 149-163.

Fierro, R., & Lewis, F. L. (1997b). Robust practical point stabilization of a nonholonomic mobile robot using neural networks. *Journal of Intelligent & Robotic Systems*, *20*(2-4), 295-317.

Freeland, R. (1990). Instrumentation for tractor pitch and roll angle measurement. *Applied Engineering in Agriculture*, *6*(5), 548-552.

Freeland, R., Tompkins, F., & Wilhelm, L. (1989). Portable instrumentation to study performance of lawn and garden ride-on tractors. *Applied Engineering in Agriculture*, *5*(2), 143-147.

Fukao, T., Nakagawa, H., & Adachi, N. (2000). Adaptive tracking control of a nonholonomic mobile robot. *IEEE Transactions on Robotics and Automation*, *16*(5), 609-615.

Geng, Y., Cole, A., Dempster, A., Rizos, C., & Wang, J. (2007). Developing a low-cost MEMS IMU/ DGPS integrated system for robust machine automation. In *Proceedings of the 20th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2007), Fort Worth, TX* (p. 1618-1624).

Gomez-Gil, J., Ryu, J. C., Alonso-Garcia, S., & Agrawal, S. K. (2011). Development and validation of globally asymptotically stable control laws for automatic tractor guidance. *Applied Engineering in Agriculture*, *27*(6), 1099-1108.

Guo, L., Zhang, Q., & Feng, L. (2003). A low-cost integrated positioning system of GPS and inertial sensors for autonomous agricultural vehicles. In *ASAE Anual International Meeting, Paper number: 033112.* St. Joseph, Mich.: ASABE.

Han, S., Zhang, Q., & Noh, H. (2004). A guidance directrix approach to vision-based vehicle guidance systems. *Computers and Electronics in Agriculture*, *43*, 179-195.

Hiremath, S. A., van der Heijden, G. W., van Evert, F. K., Stein, A., & ter Braak, C. J. (2014). Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter. *Computers and Electronics in Agriculture*, *100*(Supplement C), 41-50.

Hwang, E.-J., Kang, H.-S., Hyun, C.-H., & Park, M. (2013). Robust backstepping control based on a Lyapunov redesign for skid-steered wheeled mobile robots. *International Journal of Advanced Robotic Systems*, *10*(26), 1-8.

Indiveri, G. (1999). Kinematic time-invariant control of a 2D nonholonomic vehicle. In *Proceedings of the 38th IEEE Conference on Decision and Control. Phoenix, USA* (Vol. 3, p. 2112-2117).

Jang, J. O. (2011). Adaptive neuro-fuzzy network control for a mobile robot. *Journal of Intelligent & Robotic Systems*, *62*(3-4), 567-586.

Ji, R. H., & Qi, L. J. (2010). Crop-row detection algorithm based on random hough transformation. *Mathematical and Computer Modelling*, *54*(3-4), 1016-1020.

Jiang, Z.-P., & Nijmeijer, H. (1997). Tracking control of mobile robots: A case study in backstepping. *Automatica*, *33*(7), 1393-1399.

Kanayama, Y., Kimura, Y., Miyazaki, F., & Noguchi, T. (1991). A stable tracking control method for a non-holonomic mobile robot. In *Proceedings of IEEE/RSJ International Workshop on Intelligent Robots and Systems, IROS 1991* (Vol. 3, p. 1236-1241).

Kise, M., Zhang, Q., & Rovira Más, F. (2005). A stereovision-based crop row detection method for tractor-automated guidance. *Transactions of the ASAE*, *90*(4), 357-367.

Klančar, G., & Škrjanc, I. (2007). Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, *55*(6), 460-469.

Kurita, H., Iida, M., Cho, W., & Suguri, M. (2017). Rice autonomous harvesting: Operation framework. *Journal of Field Robotics*, *34*(6), 1084-1099.

Laguë, C., Roy, P.-M., & Savard, P. (1997). Wide-Span Implement Carrier (WSIC) for cranberry production. *Applied Engineering in Agriculture*, *13*(3), 309-317.

Lee, K. H., Ehsani, R., & Schueller, J. K. (2007). Forward movement synchronization of two vehicles in parallel using a laser scanner. *Applied Engineering in Agriculture*, *23*(6), 827-834.

Lenain, R., Deremetz, M., Braconnier, J.-B., Thuilot, B., & Rousseau, V. (2017). Robust sideslip angles observer for accurate off-road path tracking control. *Advanced Robotics*, *31*(9), 453-467.

Lenain, R., Thuilot, B., Cariou, C., & Martinet, P. (2003). Adaptive control for car like vehicles guidance relying on RTK GPS: rejection of sliding effects in agricultural applications. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2003* (Vol. 1, p. 115-120).

Lenain, R., Thuilot, B., Cariou, C., & Martinet, P. (2006). High accuracy path tracking for vehicles in presence of sliding: Application to farm vehicle automatic guidance for agricultural tasks. *Autonomous Robots*, *21*(1), 79-97.

Lenain, R., Thuilot, B., Cariou, C., & Martinet, P. (2009). Mixed kinematic and dynamic sideslip angle observer for accurate control of fast off-road mobile robots. *Journal of Field Robotics*, *27*(2), 181-196.

Li, M., Imou, K., Wakabayashi, K., & Yokoyama, S. (2009). Review of research on agricultural vehicle autonomous guidance. *International Journal of Agricultural and Biological Engineering*, *3*(2), 1-16.

Li, Y., Efatmaneshnik, M., & Dempster, A. G. (2012). Attitude determination by integration of MEMS inertial sensors and GPS for autonomous agriculture applications. *GPS Solutions*, *16*(1), 41-52.

Low, C. B., & Wang, D. (2008a). GPS-based path following control for a car-like wheeled mobile robot with skidding and slipping. *IEEE Transactions on Control Systems Technology*, *16*(2), 340-347.

Low, C. B., & Wang, D. (2008b). GPS-based tracking control for a car-like wheeled mobile robot with skidding and slipping. *IEEE/ASME Transactions on Mechatronics*, *13*(4), 480-484.

Lucet, E., Lenain, R., & Grand, C. (2015). Dynamic path tracking control of a vehicle on slippery terrain. *Control Engineering Practice*, *42*(Supplement C), 60-73.

Luo, X., & Zhang, Z. (2007). DGPS navigation control system for rice transplanter. In *ASABE Anual International Meeting, Paper number: 071013.* St. Joseph, Mich.: ASABE.

Maalouf, E., Saad, M., & Saliah, H. (2006). A higher level path tracking controller for a four-wheel differentially steered mobile robot. *Robotics and Autonomous Systems*, *54*(1), 23-33.

Meyer, S. (2010). Braking deceleration and starting acceleration of modern agricultural tractors and combine harvesters. In *19th EVU Conference, Prague.*

Mnif, F. (2004). Recursive backstepping stabilization of a wheeled mobile robot. *International Journal of Advanced Robotic Systems*, *1*(4), 287-294.

Möller, J. (2010). Computer vision – A versatile technology in automation of agriculture machinery. In *21st Annual Meeting Bologna, EIMA International.*

Monroe, G. E., & Burt, E. C. (1989). Wide frame tractive vehicle for controlled-traffic research. *Applied Engineering in Agriculture*, *5*(1), 40-43.

Morgan, K. (1958). A step towards an automatic tractor. *Farm Mech.*, *10(13)*, 440-441.

Murray, R. M., & Sastryt, S. S. (1990). Steering nonholonomic systems using sinusoids. In *Proceedings of the 29th Conference on Decision and Control. Honolulu, Hawaii* (p. 2097-2101).

Murray, R. M., & Sastryt, S. S. (1991). Steering nonholonomic systems in chained form. In *Proceedings of the 30th Conference on Decision and Control. Brighton, England* (p. 1121-1126).

Murray, R. M., & Sastryt, S. S. (1993). Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, *38*(5), 700-716.

Nagasaka, Y., Saito, H., Tamaki, K., Seki, M., Kobayashi, K., & Taniwaki, K. (2009). An autonomous rice transplanter guided by global positioning system and inertial measurement unit. *Journal of Field Robotics*, *26*(6-7), 537-548.

Nagasaka, Y., Taniwaki, K., Otani, R., & Shigeta, K. (2002). An automated rice transplanter with RTKGPS and FOG. *Agricultural Engineering International: CIGR Journal*, *4*, 1-7.

Nagasaka, Y., Umeda, N., Kanetai, Y., Taniwaki, K., & Sasaki, Y. (2004). Autonomous guidance for rice transplanting using global positioning and gyroscopes. *Computers and Electronics in Agriculture*, *43*(3), 223-234.

Noguchi, N., Will, J., Reid, J., & Zhang, Q. (2004). Development of a master-slave robot system for farm operations. *Computers and Electronics in Agriculture*, *44*(1), 1-19.

O'Connor, M., Bell, T., Elkaim, G., & Parkinson, B. (1996). Automatic steering of farm vehicles using GPS. In *3rd International Conference on Precision Agriculture.*

Oriolo, G., Luca, A. D., & Vendittelli, M. (2002). WMR control via dynamic feedback linearization: Design, implementation, and experimental validation. *IEEE Transactions on Control Systems Technology*, *10*(6), 835-852.

Park, K., Chung, H., & Lee, J. G. (2000). Point stabilization of mobile robots via state-space exact feedback linearization. *Robotics and Computer Integrated Manufacturing*, *16*(5), 353-363.

Perez-Ruiz, M., Slaughter, D. C., Gliever, C., & Upadhyaya, S. K. (2012). Automatic GPS-based intra-row weed knife control system for transplanted row crops. *Biosystems Engineering*, *111*(1), 64-71.

Pourboghrat, F., & Karlsson, M. P. (2002). Adaptive control of dynamic mobile robots with nonholonomic constraints. *Computers and Electrical Engineering*, *28*(4), 241-253.

Reid, J. F., Zhang, Q., Noguchi, N., & Dickson, M. (2000). Agricultural automatic guidance research in North America. *Computers and Electronics in Agriculture*, *25*(1-2), 155-167.

Rovira Más, F., Zhang, Q., & Hansen, A. C. (2010). *Mechatronics and intelligent systems for off-road vehicles.* New York: Springer London.

Rovira Más, F., Zhang, Q., & Reid, J. F. (2004). Automated agricultural equipment navigation using stereo disparity images. *Transactions of the ASAE*, *47*(4), 1289-1300.

Samson, C. (1995). Control of chained systems application to path following and time-varying point-stabilization of mobile robots. *IEEE Transactions on Automatic Control*, *40*(1), 64-77.

Sarkar, N., Yun, X., & Kumar, V. (1994). Control of mechanical systems with rolling constraints: Application to dynamic control of mobile robots. *The International Journal of Robotics Research*, *13*(1), 55-69.

Scaglia, G., Rosales, A., Quintero, L., Mut, V., & Agarwal, R. (2010). A linear-interpolation-based controller design for trajectory tracking of mobile robots. *Control Engineering Practice*, *18*(3), 318-329.

Schönberg, T., Ojala, M., Suomela, J., Torpo, A., & Halme, A. (1995). Positioning an autonomous off-road vehicle by using fused DGPS and inertial navigation. In *2nd IFAC Conference on Intelligent Autonomous Vehicles* (p. 226-231).

Shim, H.-S., & Sung, Y.-G. (2004). Stability and four-posture control for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, *20*(1), 148-154.

Stoll, A., & Kutzbach, D. (2000). Guidance of a forage harvester with GPS. *Precision Agriculture*, *2*, 281-291.

Subramanian, V., Burks, T. F., & Dixon, W. E. (2009). Sensor fusion using fuzzy logic enhanced Kalman filter for autonomous vehicle guidance in citrus groves. *Transactions of the ASABE*, *52*(5), 1411-1422.

Sutiarso, L., Kurosaki, H., Takigawa, T., Koike, M., Yukumoto, O., & Hasegawa, H. (2002). Trajectory control and its application to approach a target: Part II. Target approach experiments. *Transactions of the ASAE*, *45*(4), 1199-1205.

Takigawa, T., Sutiarso, L., Koike, M., Kurosaki, H., & Hasegawa, H. (2002). Trajectory control and its application to approach a target: Part I. Development of trajectory control algorithms for an autonomous vehicle. *Transactions of the ASAE*, *45*(4), 1191-1197.

Tayebi, A., & Rachid, A. (1996). Path following control law for an industrial mobile robot. In *Proceedings of the IEEE International Conference on Control Applications. Dearborn, MI* (p. 703-707).

Thuilot, B., Cariou, C., Martinet, P., & Berducat, M. (2002). Automatic guidance of a farm tractor relying on a single CP-DGPS. *Autonomous Robots*, *13*(1), 53-71.

Tillett, N. D., Holt, J. B., Chestney, A. A. W., & Reed, J. N. (1988). Experimental field gantry for leaf vegetable production: specification, design and evaluation. *Journal of Agricultural Engineering Research*, *41*, 53-64.

Tsubota, R., Noguchi, N., & Mizushima, A. (2004). Automatic guidance with laser scanner for a robot tractor in an orchard. In *Proceedings of the Conference on Automation Technology for Off-road Equipment. Kyoto, Janpan.*

Tu, X. (2013). *Robust navigation control and headland turning optimization of agricultural vehicles* (PhD Dissertation). Iowa State University, Iowa, USA.

Williford, J. R. (1980). Controlled-traffic system for cotton production. *Transactions of the ASAE*, *23*(1), 65-70.

Xue, J., Zhang, L., & Grift, T. E. (2012). Variable field-of-view machine vision based row guidance of an agricultural robot. *Computers and Electronics in Agriculture*, *84*(Supplement C), 85-91.

Yang, J. M., & Kim, J. H. (1999). Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, *15*(3), 578-587.

Yoo, S. (2012). Approximation-based adaptive control for a class of mobile robots with unknown skidding and slipping. *International Journal of Control, Automation, and Systems*, *10*(4), 703-710.

Zhang, C., Noguchi, N., & Yang, L. (2016). Leader-follower system using two robot tractors to improve work efficiency. *Computers and Electronics in Agriculture*, *121*, 269-281.

Zhang, Q., & Qiu, H. (2004). A dynamic path search algorithm for tractor automatic navigation. *Transactions of the ASAE*, *47*(2), 639-646.

Zhang, Q., Reid, J. F., & Noguchi, N. (1999). Agricultural vehicle navigation using multiple guidance sensros. In *Proceedings of International Conference on Field and Service Robotics. Pittsburgh, PA, USA* (p. 293-298).

Zhang, X., Geimer, M., Noack, P. O., & Grandl, L. (2010). A semi-autonomous tractor in an intelligent master-slave vehicle system. *Intelligent Service Robotics*, *3*(4), 263-269.

APPENDIX

Appendix A

Listings of computer programs

The main programs for the guidance system were developed on the onboard laptop in MATLAB. Besides, an Arduino program was developed on the Arduino microcontroller for the driving and steering controls of the motors. For both WSIC operating modes, a simple graphical user interface (GUI) was designed to facilitate the GPS origin setting, control executing, and data saving for the guidance system.

A.1 MATLAB programs for the mobile mode

The designed GUI for the mobile mode is shown in Figure A.1. The main functions of the GUI include setting the GPS coordinates for the origin of the local Cartesian coordinate system, choosing the robot operating mode (master or slave), and executing the automatic control procedures. After the proper values for the local origin are entered and the origin is set, the Start button is enabled to start setting up the serial communications for the IMU, GPS, XBee, and Arduino. To run the synchronous tracking test, the master robot needs to first run the program and start sending data packages through the XBee. The master robot should be kept static before the slave robot starts its program. When the slave program starts and the data communication between the two robots is established, the experimental platform enters a standby mode. Then a forward velocity can be commanded to the master, and the slave will start following the motion of the master automatically. By clicking the Stop button, the tracking process will stop and the experimental data will be saved to the laptop hard drive as Excel files (.xlsx format).

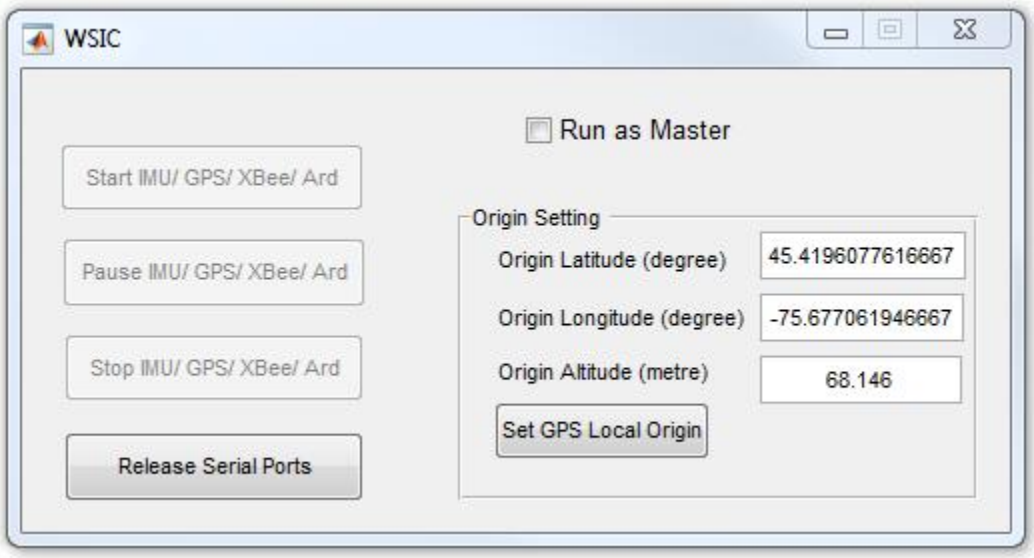


Figure A.1: GUI for the mobile operating mode.

Listing A.1: Program for the WSIC GUI of the mobile mode[1]

```
%% Function WSIC
function varargout = WSIC(varargin)

% WSIC M-file for WSIC.fig
%      WSIC, by itself, creates a new WSIC or raises the existing
%      singleton*.
%
%      H = WSIC returns the handle to a new WSIC or the handle to
%      the existing singleton*.
%
%      WSIC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in WSIC.M with the given input arguments.
%
%      WSIC('Property','Value',...) creates a new WSIC or raises the
%      existing singleton*.  Starting from the left, property value pairs
   are
%      applied to the GUI before WSIC_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
   application
%      stop.  All inputs are passed to WSIC_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WSIC

% Last Modified by GUIDE v2.5 04-Apr-2016 15:10:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @WSIC_OpeningFcn, ...
                   'gui_OutputFcn',  @WSIC_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before WSIC is made visible.
function WSIC_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WSIC (see VARARGIN)

% Choose default command line output for WSIC
handles.output = hObject;

% --- Initialization, default parameters that can be set in the GUI
global IS_PAUSED;
IS_PAUSED = false;
ori_Latitude = 0;
ori_Longitude = 0;
ori_Altitude = 0;

setappdata(hObject, 'ori_Latitude', ori_Latitude);
setappdata(hObject, 'ori_Longitude', ori_Longitude);
setappdata(hObject, 'ori_Altitude', ori_Altitude);

% Set control buttons Enable states
set(handles.btn_start_serial,'Enable','off');
set(handles.btn_pause_serial,'Enable','off');
set(handles.btn_stop_serial,'Enable','off');

% Update handles structure
guidata(hObject, handles);

% Load all libraries
load_folder_subfolder_libraries;
% UIWAIT makes WSIC wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = WSIC_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btn_start_serial.
% --- Main communication and control function
function btn_start_serial_Callback(hObject, eventdata, handles)
% hObject    handle to btn_start_serial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_PAUSED;
global IS_RUNNING;
IS_RUNNING = true;
global IS_MASTER;

global IMU_SERIAL_PORT;
global IMU_Error;
global IMU_DATA;

global GPS_SERIAL_PORT;
global GPS_Error;
global GPS_DATA;

global XBEE_SERIAL_PORT;
global XBEE_Error;
global CONTROL_DATA;

global ARD_SERIAL_PORT;
global ARD_Error;

global GPS_ORIGIN;

ori_Latitude = getappdata(handles.figure1, 'ori_Latitude');
ori_Longitude = getappdata(handles.figure1, 'ori_Longitude');
ori_Altitude = getappdata(handles.figure1, 'ori_Altitude');

GPS_ORIGIN.latitude = ori_Latitude;
GPS_ORIGIN.longitude = ori_Longitude;
GPS_ORIGIN.altitude = ori_Altitude;

% Start setting up serial communications, if stopped before (not paused)
% COM port numbers depend on physical connections, check in Device Manager
if (~IS_PAUSED)
    [IMU_SERIAL_PORT, IMU_Error] = setup_imu(6);
    [GPS_SERIAL_PORT, GPS_Error] = setup_gps(38);
    [XBEE_SERIAL_PORT, XBEE_Error] = setup_xbee(34);
    [ARD_SERIAL_PORT, ARD_Error] = setup_arduino(33);
end

if (~GPS_Error)
    gps_read_error = false;
    if ~IS_PAUSED
```

```
        % Start GPS data reading and saving
        [gps_first_result, gps_read_error] = read_gps(GPS_SERIAL_PORT,
            GPS_ORIGIN);
        GPS_DATA = [fieldnames(gps_first_result)'; struct2cell(
            gps_first_result)'];
    end

    if (~gps_read_error && ~IMU_Error && ~XBEE_Error && ~ARD_Error)
        % Start IMU data reading and saving
        [imu_first_packet, ~] = read_imu_EulerAngles(IMU_SERIAL_PORT);
        IMU_DATA = [fieldnames(imu_first_packet)'; struct2cell(
            imu_first_packet)'];

        % Start control values sending/receiving and saving
        if (IS_MASTER)
            control_packet_first = send_packets(XBEE_SERIAL_PORT,
                ARD_SERIAL_PORT, gps_first_result, imu_first_packet);
            CONTROL_DATA = [fieldnames(control_packet_first)'; struct2cell(
                control_packet_first)'];
        else
            control_packet_first = receive_packets(XBEE_SERIAL_PORT,
                ARD_SERIAL_PORT, gps_first_result, imu_first_packet);
            CONTROL_DATA = [fieldnames(control_packet_first)'; struct2cell(
                control_packet_first)'];
        end

        % Set control buttons Enable states accordingly
        set(handles.btn_start_serial,'Enable','off');
        set(handles.btn_pause_serial,'Enable','on');
        set(handles.btn_stop_serial,'Enable','on');
    else
        GPS_Error = true;
    end
end

while (IS_RUNNING && ~GPS_Error && ~XBEE_Error && ~IMU_Error && ~ARD_Error)
    [gps_data_ENU, gps_read_error] = read_gps(GPS_SERIAL_PORT, GPS_ORIGIN);
    if (~gps_read_error)
        if (~isempty(gps_data_ENU))
            GPS_DATA = [GPS_DATA; struct2cell(gps_data_ENU)'];
        end
    end
    % The pause is needed to not block the GUI!
    pause(0.00001);

    [imu_packet, imu_read_error] = read_imu_EulerAngles(IMU_SERIAL_PORT);
    if (~imu_read_error)
        IMU_DATA = [IMU_DATA; struct2cell(imu_packet)'];
    end
    pause(0.00001);

    if (~gps_read_error && ~imu_read_error)
        if IS_MASTER % if Master, send packets to Slave
            [control_packet, control_error] = send_packets(XBEE_SERIAL_PORT,
                ARD_SERIAL_PORT, gps_data_ENU, imu_packet);
        else % if Slave, receive packets from Master
```

```
            [control_packet, control_error] = receive_packets(
                XBEE_SERIAL_PORT, ARD_SERIAL_PORT, gps_data_ENU, imu_packet);
        end
        if (~control_error)
            CONTROL_DATA = [CONTROL_DATA; struct2cell(control_packet)'];
        end
    end
    pause(0.00001);
end

% Write data to Excel files, close and delete serial ports
if (~IS_PAUSED && ~GPS_Error && ~XBEE_Error && ~ARD_Error && ~IMU_Error)
    fclose(IMU_SERIAL_PORT);
    delete(IMU_SERIAL_PORT);
    fclose(GPS_SERIAL_PORT);
    delete(GPS_SERIAL_PORT);
    fclose(XBEE_SERIAL_PORT);
    delete(XBEE_SERIAL_PORT);
    fclose(ARD_SERIAL_PORT);
    delete(ARD_SERIAL_PORT);

    xlswrite('G:\results_gps.xlsx', GPS_DATA);
    xlswrite('G:\results_imu.xlsx', IMU_DATA);
    xlswrite('G:\results_control.xlsx', CONTROL_DATA);
end

% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in btn_pause_serial.
function btn_pause_serial_Callback(hObject, eventdata, handles)
% hObject    handle to btn_pause_serial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_RUNNING;
global IS_PAUSED;
IS_RUNNING = false;
IS_PAUSED = true;

set(handles.btn_start_serial,'Enable','on');
set(handles.btn_pause_serial,'Enable','off');
set(handles.btn_stop_serial,'Enable','on');


% --- Executes on button press in btn_stop_serial.
function btn_stop_serial_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_stop_gps (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_PAUSED;
global IS_RUNNING;
IS_PAUSED = false;
IS_RUNNING = false;
```

```
set(handles.btn_start_serial,'Enable','on');
set(handles.btn_pause_serial,'Enable','off');
set(handles.btn_stop_serial,'Enable','off');


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

clear global;
pause(1);
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes on button press in checkbox_master.
function checkbox_master_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_master (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_master
global IS_MASTER;
if get(hObject,'Value')
    IS_MASTER = true;
else
    IS_MASTER = false;
end


function edit_GpsLat_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsLat_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsLat_Ori as text
%        str2double(get(hObject,'String')) returns contents of
   edit_GpsLat_Ori as a double
edit_txt_content = str2double(get(hObject,'String'));
if isnan(edit_txt_content)
    msgbox('The Latitude should be a double value in degrees!');
else
    if (edit_txt_content >= 0 && edit_txt_content <= 90)
        ori_Latitude = edit_txt_content;
    else
        msgbox(['The Latitude should be between [' num2str(0) ',' ...
                num2str(90) '] degrees!']);
    end
end
set(hObject,'String',sprintf('%0.13f', ori_Latitude));

% --- Executes during object creation, after setting all properties.
function edit_GpsLat_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsLat_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','45.4196077616667'); % default latitude for the origin


function edit_GpsLon_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsLon_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsLon_Ori as text
%        str2double(get(hObject,'String')) returns contents of
    edit_GpsLon_Ori as a double
edit_txt_content = str2double(get(hObject,'String'));
if isnan(edit_txt_content)
    msgbox('The Longitude should be a double value in degrees!');
else
    if (edit_txt_content >= -180 && edit_txt_content <= 180)
        ori_Longitude = edit_txt_content;
    else
        msgbox(['The Longitude should be between (' num2str(-180) ',' ...
                num2str(180) '] degrees!']);
    end
end
set(hObject,'String',sprintf('%0.13f', ori_Longitude));

% --- Executes during object creation, after setting all properties.
function edit_GpsLon_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsLon_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','-75.677061946667'); % default longitude for the
    origin


function edit_GpsAlt_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsAlt_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsAlt_Ori as text
%        str2double(get(hObject,'String')) returns contents of
    edit_GpsAlt_Ori as a double
edit_txt_content = str2double(get(hObject,'String'));
```

```
if isnan(edit_txt_content)
    msgbox('The Altitude should be a double value in metres!');
else
    ori_Altitude = edit_txt_content;
end
% Uses up to 3 floating points to show values in milimeters precision
set(hObject,'String',sprintf('%0.3f', ori_Altitude));

% --- Executes during object creation, after setting all properties.
function edit_GpsAlt_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsAlt_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','68.146'); % default altitude for the origin


% --- Executes on button press in btn_release_ports.
function btn_release_ports_Callback(hObject, eventdata, handles)
% hObject    handle to btn_release_ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(instrfindall);


% --- Executes on button press in btn_set_origin.
function btn_set_origin_Callback(hObject, eventdata, handles)
% hObject    handle to btn_set_origin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject, 'value')
    % Read origin setting values from text boxes
    ori_lat = str2double(get(handles.edit_GpsLat_Ori, 'String'));
    ori_lon = str2double(get(handles.edit_GpsLon_Ori, 'String'));
    ori_alt = str2double(get(handles.edit_GpsAlt_Ori, 'String'));

    setappdata(handles.figure1, 'ori_Latitude', ori_lat);
    setappdata(handles.figure1, 'ori_Longitude', ori_lon);
    setappdata(handles.figure1, 'ori_Altitude', ori_alt);

    % When the origin is set, enable the "Start" button
    set(handles.btn_start_serial,'Enable','on');
end
```

[1]Function `WSIC.m` is one single file for the GUI.

Listing A.2: Programs for the IMU[2]

```
%% Function setup_imu
function [imu_serial_port, Error] = setup_imu(device_port)

Error = false;
imu_serial_port = '';
delete(instrfind);

PROMPT01 = 'Please enter COM port # (1 for COM1, etc.) for  IMU';
TITLE01 = 'COM port';
% Input the serial port Number for the IMU
ComNum = char(inputdlg(PROMPT01, TITLE01, 1, {num2str(device_port)}));

if isstrprop(ComNum, 'digit')
    imu_serial_port = instrfind('Type', 'serial', 'Port', strcat('COM',
       ComNum), 'Tag', '');
    if isempty(imu_serial_port)
        imu_serial_port = serial(strcat('COM', ComNum));
    else
        fclose(imu_serial_port);
        imu_serial_port = imu_serial_port(1);
    end
    % Set serial port parameters
    imu_serial_port.InputBufferSize = 512;
    imu_serial_port.OutputBufferSize = 512;
    imu_serial_port.BaudRate = 115200;
    imu_serial_port.DataBits = 8;
    imu_serial_port.StopBit = 1;
    imu_serial_port.Timeout = 1/20.1;
    imu_serial_port.Parity = 'none';
    imu_serial_port.BytesAvailableFcnMode = 'terminator';
    imu_serial_port.Terminator = 'CR/LF';

    try
        fopen(imu_serial_port);
    catch exception
        fprintf(['\n' exception.message '\n']);
        Error = true;
    end
else
    fprintf('\nInvalid COM port selected\n');
    Error = true;
end


%% Function read_imu_EulerAngles
function [imu_packet, error] = read_imu_EulerAngles(imu_serial_port)

imu_packet = '';
error = false;

try
        % Read IMU data strings into a.
        a = dec2hex((fread(imu_serial_port, imu_serial_port.BytesAvailable,
            'uchar'))');
        % b is the periodic string header
```

```
        b = ['75'; '65'; '80'; '0E'; '0E'; '0C'];
        [~,Locb] = ismember(b,a, 'rows'); % get the location of b in a
        if Locb(2)-Locb(1)==1 && Locb(3)-Locb(2)==1 && Locb(4)-Locb(3)==1
        n = Locb(1);
        roll_hex = [a(n+6,:) a(n+7,:) a(n+8,:) a(n+9,:)];
        pitch_hex = [a(n+10,:) a(n+11,:) a(n+12,:) a(n+13,:)];
        yaw_hex = [a(n+14,:) a(n+15,:) a(n+16,:) a(n+17,:)];
        end
    % Convert radians to degrees
        imu_packet.roll = rad2deg(hexsingle2num(roll_hex));
        imu_packet.pitch = rad2deg(hexsingle2num(pitch_hex));
        imu_packet.yaw = rad2deg(hexsingle2num(yaw_hex));
catch exception
        fprintf(['\n' exception.message '\n']);
        error = true;
end


%% Function hexsingle2num
% Source code from https://www.mathworks.com/matlabcentral/fileexchange
   /6927-hexsingle2num
function x = hexsingle2num(s)
% HEXSINGLE2NUM Convert single precision IEEE hexadecimal string to number.
%   HEXSINGLE2NUM(S), where S is a 8 character string containing
%   a hexadecimal number, returns a double type number
%   equal to the IEEE single precision
%   floating point number it represents.  Fewer than 8
%   characters are padded on the right with zeros.
%
%   If S is a character array, each row is interpreted as a single
%   precision number (and returned as a double).
%
%   NaNs, infinities and denorms are handled correctly.
%
%   Example:
%       hexsingle2num('40490fdb') returns Pi.
%       hexsingle2num('bf8') returns -1.
%
%   See also HEX2NUM.

% Based on Matlab's hex2num.
% Note: IEEE Standard 754 for floating point numbers
%
%  Floating point numbers are represented as:
%  x = +/- (1+f)*2^e
%
%  doubles: 64 bits
%           Bit 63       (1 bit)  = sign (0=positive, 1=negative)
%           Bit 62 to 52 (11 bits)= exponent biased by 1023
%           Bit 51 to 0  (52 bits)= fraction f of the number 1.f
%  singles: 32 bits
%           Bit 31       (1 bit)  = sign (0=positive, 1=negative)
%           Bit 30 to 23 (8 bits) = exponent biased by 127
%           Bit 22 to 0  (23 bits)= fraction f of the number 1.f

% 21 June 2005 Fixed bug with underflow.
%    Bug found by Matthias Noell (matthias.noell@heidelberg.com)
```

```
if iscellstr(s), s = char(s); end
if ~ischar(s)
    error('Input to hexsingle2num must be a string.')
end
if isempty(s), x = []; return, end

[row,col] = size(s);
blanks = find(s==' '); % Find the blanks at the end
if ~isempty(blanks), s(blanks) = '0'; end % Zero pad the shorter hex
   numbers.

% Convert characters to numeric digits.
% More than 8 characters are ignored
% For double: d = zeros(row,16);
d = zeros(row,8);
d(:,1:col) = abs(lower(s)) - '0';
d = d + ('0'+10-'a').*(d>9);
neg = d(:,1) > 7;
d(:,1) = d(:,1)-8*neg;

if any(d > 15) | any(d < 0)
    error('Input string to hexsingle2num should have just 0-9, a-f, or A-F.
       ')
end

% Floating point exponent.
% For double: e = 16*(16*(d(:,1)-4) + d(:,2)) + d(:,3) + 1;
% For double: e = 256*d(:,1) + 16*d(:,2) + d(:,3) - 1023;
expBit = (d(:,3) > 7);
e = 32*d(:,1) + 2*d(:,2) + expBit - 127;
d(:,3) = d(:,3)-8*expBit;  % Remove most sig. bit of d(:,3) which belongs
   to exponent

% Floating point fraction.
% For double: sixteens = [16;256;4096;65536;1048576;16777216;268435456];
% For double: sixteens2 = 268435456*sixteens(1:6);
% For double: multiplier = 1./[sixteens;sixteens2];
% For double: f = d(:,4:16)*multiplier;
sixteens = [16;256;4096;65536;1048576;16777216];
multiplier = 2./[sixteens];
f = d(:,3:8)*multiplier;

x = zeros(row,1);
% Scale the fraction by 2 to the exponent.
% For double: overinf = find((e>1023) & (f==0));
overinf = find((e>127) & (f==0));
if ~isempty(overinf), x(overinf) = inf; end

% For double: overNaN = find((e>1023) & (f~=0));
overNaN = find((e>127) & (f~=0));
if ~isempty(overNaN), x(overNaN) = NaN; end

% For double: underflow = find(e<-1022);
underflow = find(e<-126);
if ~isempty(underflow), x(underflow) = pow2(f(underflow),-126); end
```

```
% For double: allothers = find((e<=1023) & (e>=-1022));
allothers = find((e<=127) & (e>=-126));
if ~isempty(allothers), x(allothers) = pow2(1+f(allothers),e(allothers));
    end

negatives = find(neg);
if ~isempty(negatives), x(negatives) = -x(negatives); end
```

[2]Programs for the IMU include three separate files: `setup_imu.m`, `read_imu_EulerAngles.m`, and `hexsingle2num.m`.

Listing A.3: Programs for the GPS[3]

```
%% Function setup_gps
function [gps_serial_port, Error] = setup_gps(device_port)

Error = false;
gps_serial_port = '';

PROMPT01 = 'Please enter COM port # (1 for COM1, etc.) for  GPS';
TITLE01 = 'COM port';
% Input the serial port Number for the GPS
ComNum = char(inputdlg(PROMPT01, TITLE01, 1, {num2str(device_port)}));

if isstrprop(ComNum, 'digit')
    gps_serial_port = instrfind('Type', 'serial', 'Port', strcat('COM',
       ComNum), 'Tag', '');
    if isempty(gps_serial_port)
        gps_serial_port = serial(strcat('COM', ComNum));
    else
        fclose(gps_serial_port);
        gps_serial_port = gps_serial_port(1);
    end
    % Set serial port parameters
    gps_serial_port.InputBufferSize = 512;
    gps_serial_port.OutputBufferSize = 512;
    gps_serial_port.BaudRate = 115200;
    gps_serial_port.DataBits = 8;
    gps_serial_port.StopBit = 1;
    gps_serial_port.Timeout = 5;
    gps_serial_port.Parity = 'none';
    gps_serial_port.BytesAvailableFcnMode = 'terminator';
    gps_serial_port.Terminator = 'CR/LF';

    try
        fopen(gps_serial_port);
    catch exception
        fprintf(['\n' exception.message '\n']);
        Error = true;
    end
else
    fprintf('\nInvalid COM port selected\n');
    Error = true;
end
```

```
%% Function read_gps
function [result_data_ENU, error] = read_gps(gps_serial_port, gps_origin)
% Reads NMEA GPGGA and GPGMC sentences and returns East-North-Up
% (ENU) coordinates and speeds. This function calls nmealineread
% and convert2enu.

result_data_ENU = '';
error = false;

try
    data = fscanf(gps_serial_port);
    % Read the GPGGA sentences to get latitudes and longitudes
```

```
    if (strncmp('$GPGGA',data, 6) == 1)
        [result_data, error] = nmealineread(data);
        if (error ~= -1)
            result_data_ENU = convert2enu(result_data, gps_origin);
            % Read the GPRMC sentences to get speeds.
            data = fscanf(gps_serial_port);
            [result_data_GPRMC, error] = nmealineread(data);
            result_data_ENU.speed = convvel(result_data_GPRMC.groundspeed.
               knot,'kts','m/s');
        end
    end
catch exception
    fprintf(exception.message);
    error = true;
end
```

```
%% Function nmealineread
% Source code from https://www.mathworks.com/matlabcentral/fileexchange
   /45616-improved-nmea-line-reader
function [data,ierr]  =  nmealineread(nline)
% NMEALINEREAD reads an NMEA sentence into a MATLAB structure array
%
%  DATA  =  NMEALINEREAD(NLINE)
%  [DATA,IERR]  =  NMEALINEREAD(NLINE)
%
%  NLINE is an NMEA sentence. DATA is a MATLAB structure array with a
%  varying format, detailed below.
%
%  NMEALINEREAD currently supports the following NMEA sentences:
%                  $GPGGA  Global positioning system fixed data
%                  $GPGLL  Geographic poition [latitude, longitude & time]
%                  $GPVTG  Course over ground and ground speed
%                  $GPZDA  UTC date / time and local time zone offset
%                  $SDDBS  Echo sounder data
%
%
%  IERR returns an error code:
%       -2  -  NMEA string recognised, but function not yet able to read
%              this string
%       -1  -  NMEA string not recognised
%       0   -  No errors

% Adam Leadbetter (alead@bodc.ac.uk) -      2006-Oct-24
% Lex Lombardi (llombardi@dspaceinc.com) -  2014-Feb-19
%                       partially updated to use textscan() for parsing
%                       added support for other fields in GPGGA
%                       added checksum calculator

ierr  =  0;

% Set up a list of valid NMEA strings
nmea_options  =  [  '$GPGGA'
                    '$GPGLL'
                    '$GPGSA'
                    '$GPGSV'
                    '$GPRMC'
```

```
                    '$GPVTG'
                    '$GPZDA'
                    '$SDDBS'];
% Find which string we're dealing with
fields = textscan(nline,'%s','delimiter',',');
% Pull the checksum out of the last field and make a new one for it
fields{1}{end+1} = fields{1}{end}(end-1:end);
% Cut off the old last field at the chksum delimiter
fields{1}(end-1) = strtok(fields{1}(end-1), '*');
case_t = find(strcmp(fields{1}(1), nmea_options),1);
fields = char(fields{1});

%%%  If no valid NMEA string found - quit with an error
if isempty(case_t)
    fprintf(1,'\n\tWarning: Not a valid NMEA string  -  %s ...\n',nline);
    data  =  NaN;
    ierr  =  -1;
    return
end

% Read and check the checksum
% Initialise checksum
checksum = uint8(0);

% Calc it - we drop the leading '$' and trim off the '*' and anything past
   it
for i_char = 2:(find(nline=='*',1,'last')-1)
    checksum = bitxor(checksum, uint8(nline(i_char)));
end
checksum = dec2hex(checksum, 2);

% Check it
if ((strcmp(fields(end,1:2),checksum))==0)
   % Checksum is bad!
    fprintf(1,'\n\tWarning: Checksum Bad - %s ~= %s',fields(end),checksum);
    data  =  NaN;
    ierr  =  -1;
    return
end

%%% TURN ON THE SWITCH!

switch case_t
    case 1 %% GPGGA: Read global positioning system fixed data

        % first data field is the time
        t_time  =  fields(2,1:end);
        if(isempty(t_time))
                    % data.BODCTime = NaN;
            data.Time = NaN;
        else
            data.Time = t_time;
            % data.BODCTime = datenum(t_time,'HHMMSS') - ...
            % floor(datenum(t_time,'HHMMSS'));
        end
        clear t_time;
```

```
% next data field is the lat
t_lat  =  fields(3,1:end);
data.latitude  =  ...
    str2double(t_lat(1:2)) + (str2double(t_lat(3:end))/60);
t_latDir = strtrim(fields(4,1:end));
if(t_latDir  ==  'S')
    data.latitude  =  data.latitude  *  -1;
end
clear t_lat t_latDir;

% then the lon
t_lon  = fields(5,1:end);
data.longitude  =  ...
    str2double(t_lon(1:3)) + (str2double(t_lon(4:end))/60);
t_lonDir = strtrim(fields(6,1:end));
if(t_lonDir  ==  'W')
    data.longitude  =  data.longitude  *  -1;
end
clear t_lon t_longDir;

% get the fix quality where 0 = none, 1 = GPS fix, 2 = DGPS fix
t_fix = fields(7,1:end);
data.fix  = str2double(t_fix);
clear t_fix;

% read the number of satellites
t_sat = fields(8,1:end);
data.satellites = str2double(t_sat);
clear t_sat;

% read HDOP
t_HDOP= fields(9,1:end);
if isempty(t_HDOP)
    % do nothing
else
    data.HDOP = str2double(t_HDOP);
end
clear t_HDOP;

% Read Altitude
t_alt = fields(10,1:end);
if isempty(t_alt)
    % do nothing
else
    data.altitude = str2double(t_alt);
end
clear t_alt;

t_altUnit = fields(11,1:end);
if (t_altUnit(1)=='M')
    % do nothing
else
    fprintf(1,'\tWarning: unknown Altitude Unit - %s\n', t_altUnit)
       ;
end
clear t_altUnit;
```

```
    % Height of geoid (mean sea level)
    t_altGeo = fields(12,1:end);
    clear t_altGeo;
    t_altGeoUnit = fields(13,1:end);
    if (t_altGeoUnit(1)=='M')
        % do nothing
    else
        fprintf(1,'\tWarning: unknown Height over WGS84 Unit - %s\n',
           t_altGeoUnit);
    end
    clear t_altGeoUnit;

    % Time since DGPS update
    t_DGPSupdate = fields(14,1:end);

    % Checksum
    t_chkSum = fields(15,1:end);

case 2 %% GPGLL: Read geographic position [lat/lon] and time

    t_lat  = fields(2,1:end);
    data.latitude  =  str2double(t_lat(1:2)) + ...
        (str2double(t_lat(3:end)) / 60);
    t_latDir  =  strtrim(fields(3,1:end));
    if(t_latDir  ==  'S')
        data.latitude  =  data.latitude * -1;
    end
    clear t_lat t_latDir

    t_lon  =  fields(4,1:end);
    data.longitude  =  str2double(t_lon(1:3)) + ...
        (str2double(t_lon(4:end)) / 60);
    t_lonDir = strtrim(fields(5,1:end));
    if(t_lonDir  ==  'W')
        data.longitude  =  data.longitude  *  -1;
    end

    if(length(fields) == 7)
        t_time  =  fields(6,1:end);
        data.BODCTime  =  datenum(t_time,'HHMMSS') - ...
            floor(datenum(t_time,'HHMMSS'));
    else
        data.BODCTime  =  NaN;
    end

case 3 %% GPGSA: Read precision and fix quality information
    % fix and mode info in fields 2&3

    t_mode = fields(2,1:end);
    switch t_mode(1)
        case 'M'
            data.fixmode='Manual';
        case 'A'
            data.fixmode='Automatic';
        otherwise
            data.fixmode=NaN;
    end
```

```
        clear t_mode

        t_mode = fields(3,1:end);
        switch t_mode(1)
            case '1'
                data.fixtype=1; % no fix
                data.fix=0;
            case '2'
                data.fixtype=2; % 2D fix
                data.fix=0;
            case '3'
                data.fixmode=3; % 3D fix
                data.fix=0;
            otherwise
                data.fixmode=NaN;
        end
        clear t_mode

        % satalite id's in fields 4-15
        t_satID=str2double(fields(4:15,1:end));
        if not(isempty(t_satID))
            data.satellites=t_satID;
        else
            data.satellites=NaN;
        end

        % dilution of precision
        t_PDOP = fields(16,1:end);
        if not(isempty(t_PDOP))
            data.PDOP=str2double(t_PDOP);
        else
            data.PDOP=NaN;
        end

        t_HDOP = fields(17,1:end);
        if not(isempty(t_PDOP))
            data.HDOP=str2double(t_HDOP);
        else
            data.HDOP=NaN;
        end

        t_VDOP = fields(18,1:end);
        if not(isempty(t_VDOP))
            data.VDOP=str2double(t_VDOP);
        else
            data.VDOP=NaN;
        end
        clear t_PDOP t_HDOP t_VDOP

    case 5 %% GPRMC: Recommended minimum specific GPS/Transit data
        % first data field is the time
        t_time  =  fields(2,1:end);
        if(isempty(t_time))
            % data.BODCTime  =  NaN;
            data.Time  =  NaN;
        else
            % data.BODCTime  =  datenum(t_time,'HHMMSS') - ...
```

```
    %     floor(datenum(t_time,'HHMMSS'));
    data.Time  =  t_time;
end
clear t_time;

% second data field is the validity flag
t_validity  =  strtrim(fields(3,1:end));
if(isempty(t_validity) || ~sum(strcmp(t_validity, {'A', 'V'})))
    data.valid  =  NaN;
else
    data.valid  =  t_validity(1)=='A';
end
clear t_validity;

% third to sixth data fields are lat/lon coordinates
t_lat  = fields(4,1:end);
data.latitude  =  str2double(t_lat(1:2)) + ...
    (str2double(t_lat(3:end)) / 60);
t_latDir  =  strtrim(fields(5,1:end));
if(t_latDir  ==  'S')
    data.latitude  =  data.latitude * -1;
end
clear t_lat t_latDir

t_lon  =  fields(6,1:end);
data.longitude  =  str2double(t_lon(1:3)) + ...
    (str2double(t_lon(4:end)) / 60);
t_lonDir = strtrim(fields(7,1:end));
if(t_lonDir  ==  'W')
    data.longitude  =  data.longitude  *  -1;
end
clear t_lon t_lonDir

% seventh data field is speed in knots
t_gspeed  =  fields(8,1:end);
if(isempty(t_gspeed))
    data.groundspeed.knot  =  NaN;
else
    data.groundspeed.knot  =  str2double(t_gspeed);
end
clear t_gspeed

% eighth data field is the true course
t_course  =  fields(9,1:end);
if(isempty(t_course))
    data.truecourse  =  NaN;
else
    data.truecourse  =  str2double(t_course);
end
clear t_course

% nineth data field is the date
t_date  =  fields(10,1:end);
if(~isempty(t_date))
    data.Date = t_date;
else
    data.Date = NaN;
```

```
    end
    clear t_date

    % tenth data field is the magnetic variation degree
    t_magneticvariation_degree =  fields(11,1:end);
    if(isempty(t_magneticvariation_degree))
        data.magneticvariation.degree  =  NaN;
    else
        data.magneticvariation.degree  =  str2double(
           t_magneticvariation_degree);
    end
    clear t_magneticvariation_degree

    % eleventh data field is the magnetic variation direction
    t_magneticvariation_direction =  strtrim(fields(12,1:end));
    if(isempty(t_magneticvariation_direction) || ~sum(strcmp(
       t_magneticvariation_direction, {'E', 'W'})))
        data.magneticvariation.direction  =  NaN;
    else
        data.magneticvariation.direction  =
           t_magneticvariation_direction;
    end
    clear t_magneticvariation_direction

case 6 %% GPVTG: Read course over ground and ground speed

    t_course  =  fields(2,1:end);
    if(isempty(t_course))
        data.truecourse  =  NaN;
    else
        data.truecourse  =  str2double(t_course);
    end

    t_course  =  fields(4,1:end);
    if(isempty(t_course))
        data.magneticcourse  =  NaN;
    else
        data.magneticcourse  =  str2double(t_course);
    end

    t_gspeed  =  fields(6,1:end);
    if(isempty(t_gspeed))
        data.groundspeed.knot  =  NaN;
    else
        data.groundspeed.knot  =  str2double(t_gspeed);
    end

    t_gspeed  =  fields(6,1:end);
    if(isempty(t_gspeed))
        data.groundspeed.kph  =  NaN;
    else
        data.groundspeed.kph  =  str2double(t_gspeed);
    end

    clear t_course t_gspeed;

case 7 %%  Read UTC Date / Time and Local Time Zone Offset
```

```
        data.BODCTime  =  (datenum(nline(11:20),'dd,mm,yyyy') + ...
            (datenum(nline(1:6),'HHMMSS') - ...
            floor(datenum(nline(1:6),'HHMMSS'))));
        data.offset  =  (str2double(nline(22:23)) + ...
            (str2double(nline(25:26)) / 60)) / 24;

        case 8 %%  Read echo sounder data

        com_mask  =  strfind(nline,',');
        data.depth  =  str2double(...
        nline(com_mask(2) + 1: com_mask(3)-1));

        otherwise
        data  =  NaN;
        ierr  =  -2;
        fprintf(1,...
            '\n\tWarning: NMEA reader not yet implemented for this string
               -  %s  ...\n',...
            nline);
end

%  Tidy up the output structure
data  =  orderfields(data);


%% Function convert2enu
function [enu_result]  =  convert2enu(gps_result, gps_origin)
% CONVERT2ENU Converts the latitude, longitude, and altitude into local
% x, y, z coordinates. The values depend on the position of the origin.

referenceEllipsoid = wgs84Ellipsoid;

[x, y, z] = geodetic2enu(gps_result.latitude, gps_result.longitude, ...
    gps_result.altitude, gps_origin.latitude, ...
    gps_origin.longitude, gps_origin.altitude, referenceEllipsoid);

enu_result = gps_result;
enu_result.x = x;
enu_result.y = y;
enu_result.z = z;
```

[3]Programs for the GPS include four separate files: `setup_gps.m`, `read_gps.m`, `nmealineread.m`, and `convert2enu.m`.

Listing A.4: Programs for the XBee[4]

```
%% Function setup_xbee
function [ xbee_serial_port, Error ] = setup_xbee ( device_port )

Error = false;
xbee_serial_port = '';

PROMPT01 = ['Please enter COM port # (1 for COM1, etc.) for XBee'];
TITLE01 = 'COM port';
% Input the serial port Number for the XBee
ComNum = char(inputdlg(PROMPT01, TITLE01, 1, {num2str(device_port)}));

if isstrprop(ComNum, 'digit')
    xbee_serial_port = instrfind('Type', 'serial', 'Port', strcat('COM',
       ComNum), 'Tag', '');
    if isempty(xbee_serial_port)
        xbee_serial_port = serial(strcat('COM', ComNum));
    else
        fclose(xbee_serial_port);
        xbee_serial_port = xbee_serial_port(1);
    end
    % Set serial port parameters
    xbee_serial_port.InputBufferSize = 512;
    xbee_serial_port.OutputBufferSize = 512;
    xbee_serial_port.BaudRate = 9600;
    xbee_serial_port.DataBits = 8;
    xbee_serial_port.StopBit = 1;
    xbee_serial_port.Timeout = 5;
    xbee_serial_port.Parity = 'none';
    xbee_serial_port.BytesAvailableFcnMode = 'terminator';
    xbee_serial_port.Terminator = 'CR/LF';

    try
        fopen(xbee_serial_port);
    catch exception
        fprintf(['\n' exception.message '\n']);
        Error = true;
    end
else
    fprintf('\nInvalid COM port  for XBee selected\n');
    Error = true;
end
```

```
%% Function send_packets
function [control_packet, error] = send_packets(xbee_serial_port,
   ard_serial_port, gps_packet, imu_packet)
% send_packets.m is a function written for the Master to send data
% packets to the XBee port. The control algorithm equaiton is processed
% and control signals are sent to the Arduino serial port.

% This function extracts the local coordinates x, y, and the speed from
% the GPS packet data, and the yaw angle form IMU packets to calculate the
% steering angle for the Master vehicle. Then control signals are sent to
% the Arduino board for motor control.
```

```
%%% Define the desired straight paths by A, B and C parameters Ax+By+C = 0
% PATH1 y = 0.5997x + 4.2276
PATH1 = [-0.5997 1 -4.2276]; % PATH = [A, B, C];
% PATH2 y = 0.5997x - 3.351634
PATH2 = [-0.5997 1 3.351634]; % PATH = [A, B, C];

% Constant used in the delta formula
L = 0.495; % robot wheelbase 495 mm

error = false;
% Variables for data saving
control_packet.Delta_Master = 0;
control_packet.Speed_Master = 0;
control_packet.Orientation_Master = 0;
control_packet.Distance_Master = 0;
control_packet.Theta_p_Master = 0;
control_packet.Date = '';

try
    % Get data from GPS
    x_local = gps_packet.x;
    y_local = gps_packet.y;
    speed_master_gps = gps_packet.speed;

    % Set reference velocity
    speed_ref = 0.3; % constant reference velocity
    % curr_time = clock;
    % speed_ref = 0.4+0.2*sin(curr_time(6)*4*pi/60); % varing reference
       velocity

    % Get yaw angle from the IMU, [-180, 180]
    % North = 0, East = 90, West = -90 degree
    yaw_master = imu_packet.yaw;
    % Calibrate the orientation angle w.r.t. the straight line of slope
    % 30.951 degrees, the true yaw angle should be 59.049 degrees
    yaw_master = yaw_master -51.06 + 59.049;
    % Orientation angle of the Master w.r.t. the East
    % East = 0, North = 90, South = -90 degree
    % Below is the corversion from Yaw angle to Orientation angle.
    if (yaw_master >=-90 && yaw_master <=180)
        orientation_master = yaw_master * (-1) + 90;
    elseif(yaw_master < -90 && yaw_master >=-180)
        orientation_master = yaw_master * (-1) - 270;
    end

    % Calculate the deviation distance and path angle
    [distance, PATH_ANGLE] = distance_point_to_line([x_local, y_local],
       PATH1);
    % Calculate the orientation error
    theta_p = orientation_master - PATH_ANGLE;

    % Calculate the control signals
    K1 = 2.4;  K2 = 1.4; % control gains should be tuned during the
       experiments, K1 = 2 * sqrt(K2)
    delta_master = atand(L * (-K1*tand(theta_p) - K2*distance/cosd(theta_p)
       ));
```

```
    % Restrict the steering angle to saturation range [-26.98, 26.98]
    if (delta_master > 26.98)
        delta_master = 26.98;
    elseif (delta_master < -26.98)
        delta_master= -26.98;
    end

    % The following conversion is needed since delta_master is the steering
    % angle for the virtual wheel in the middle of the axle, while the true
    % control angle is on the left wheel.
    if (delta_master >= 0)
        delta_master_left = atand( 0.495*tand(delta_master) / (0.495-0.18*
           tand(delta_master)) );
    elseif (delta_master < 0)
        delta_master_left = atand( 0.495*tand(delta_master) / (0.495+0.18*
           tand(delta_master)) );
    end

    % Send data packets to the Slave over XBee
    serial_write_master_xbee(xbee_serial_port, x_local, y_local,
       orientation_master, speed_master_gps);
    % Send control commands to the Arduino
    serial_write_master_arduino(ard_serial_port, delta_master_left,
       speed_ref);
    % Save data variables
    control_packet.Delta_Master = delta_master;
    control_packet.Speed_Master = speed_ref;
    control_packet.Orientation_Master = orientation_master;
    control_packet.Distance_Master = distance;
    control_packet.Theta_p_Master = theta_p;
    control_packet.Date = getCurrentClockStr;

catch exception
    fprintf(exception.message);
    error = true;
end


%% Function receive_packets
function [control_packet, error] = receive_packets(xbee_serial_port,
   ard_serial_port, gps_packet, imu_packet)
%  receive_packets.m is a function written for the Slave to recieve
%  data packets from the XBee port. The control algorithm equaiton is
%  processed and control signals are sent to the Arduino serial port.
%
%  This function extracts the data packets sent from the Master (including
%  its xy coordinates, yaw angle, and speed), calculates the
%  steering and speed control inputs for the Slave, and sends control
   signals
%  to the Arduino board for motor control.

%%% Define the desired straight path by A, B and C parameters Ax+By+C = 0
% PATH1 y = 0.5997x + 4.2276
PATH1 = [-0.5997 1 -4.2276]; % PATH = [A, B, C];
% PATH2 y = 0.5997x - 3.351634
PATH2 = [-0.5997 1 3.351634]; % PATH = [A, B, C];
```

```
% Constant used in delta formula
L = 0.495; % robot wheelbase 495 mm

error = false;
% Variables for data saving
control_packet.Delta_Slave = 0;
control_packet.Speed_Slave = 0;
control_packet.Speed_GPS_Slave = 0;
control_packet.Orientation_Slave = 0;
control_packet.Lateral_Slave = 0;
control_packet.Offset = 0;
control_packet.Theta_p_Slave = 0;
control_packet.Date = '';

try
    % Get data from GPS
    x_local_slave = gps_packet.x;
    y_local_slave = gps_packet.y;
    speed_slave_gps = gps_packet.speed;

    % Get yaw angle from the IMU sensor, [-180, 180]
    % North = 0, East = 90, West = -90 degree
    yaw_slave = imu_packet.yaw;

    % Calibrate the orientation angle w.r.t. the straight line of slope
    % 29.938 degrees, the true yaw angle should be 60.062 degrees
    yaw_slave = yaw_slave +1.88 + 60.062;

    % Orientation angle of the Slave w.r.t. the East
    % East = 0, North = 90, South = -90 degree
    % Below is the corversion from Yaw angle to Orientation angle.
    if (yaw_slave >=-90 && yaw_slave <=180)
        orientation_slave = yaw_slave * (-1) + 90;
    elseif(yaw_slave < -90 && yaw_slave >=-180)
        orientation_slave = yaw_slave * (-1) - 270;
    end

    % Read Master data from the XBee serial port
    [x_local_master, y_local_master, orientation_master, speed_master] =
       serial_read_slave_xbee(xbee_serial_port);
    % Calculate the deviation distance and path angle
    [distance_slave, PATH_ANGLE] = distance_point_to_line([x_local_slave,
       y_local_slave], PATH2);
    % Calculate the orientation errors
    theta_p_master = orientation_master - PATH_ANGLE;
    theta_p_slave = orientation_slave - PATH_ANGLE;
    % Calculate the offset error
    e_offset = offset_distance([x_local_master, y_local_master], [
       x_local_slave, y_local_slave], PATH2);
    %%% Calculate the control signals
        K1 = 2.4; K2 = 1.4; Ke = 0.2; % control gains should be tuned
           during the experiments
    % Calculate speed for the Slave
    speed_slave = (speed_master*cosd(theta_p_master) + Ke*e_offset)/cosd(
       theta_p_slave);
    % Restrict the speed output to a safe range
    if (speed_slave >0.8)
```

```
        speed_slave = 0.8;
    elseif(speed_slave < -0.8)
        speed_slave = -0.8;
    end
    % Calculate the steering angle for the Slave
    delta_slave = atand( L* (-K1*tand(theta_p_slave)-K2*distance_slave/cosd
       (theta_p_slave) ));
    % Restrict the steering angle to saturation range [-26.98, 26.98]
    if (delta_slave > 26.98)
        delta_slave = 26.98;
    elseif (delta_slave < -26.98)
        delta_slave = -26.98;
    end

    % The following conversion is needed since delta_slave is the steering
    % angle for the virtual wheel in the middle of the axle, while the true
    % control angle is on the left wheel.
    if (delta_slave >= 0)
        delta_slave_left = atand( 0.495*tand(delta_slave) / (0.495-0.18*
           tand(delta_slave)) );
    elseif (delta_slave < 0)
        delta_slave_left = -atand( 0.495*tand(-delta_slave) / (0.495+0.18*
           tand(-delta_slave)) );
    end

    % Send control commands to the Arduino
    serial_write_slave_arduino(ard_serial_port, delta_slave_left,
       speed_slave);

    % Save data variables
    control_packet.Delta_Slave = delta_slave;
    control_packet.Speed_Slave = speed_slave;
    control_packet.Speed_GPS_Slave = speed_slave_gps;
    control_packet.Orientation_Slave = orientation_slave;
    control_packet.Lateral_Slave = distance_slave;
    control_packet.Offset = e_offset;
    control_packet.Theta_p_Slave = theta_p_slave;
    control_packet.Date = getCurrentClockStr;

catch exception
    fprintf(exception.message);
    error = true;
end



%% Function serial_write_master_xbee
function [final_command] = serial_write_master_xbee(xbee_serial_port,
   x_local, y_local, orientation, speed)
% This function is written for the Master to send its xy coordinates,
% orientation angle, and speed to the Slave over the XBee.
    final_command = sprintf('%f %f %f %f', x_local, y_local, orientation,
       speed)
    fprintf(xbee_serial_port, final_command);
end
```

```
%% Function serial_read_slave_xbee
function [x_local, y_local, orientation, speed] = serial_read_slave_xbee(
   serial_port)
% This function is written for the Slave to receive data from the Master.

try
    % Get the values between two terminator characters
    serial_data = fgets(serial_port);

    % Extract the values
    fields = sscanf(serial_data,'%f');
    if (numel(fields) == 4)
        x_local = fields(1);
        y_local = fields(2);
        orientation = fields(3);
        speed = fields(4);
    end

catch exception
    fprintf(exception.message);
end
```

[4]Programs for the XBee include five separate files: `setup_xbee.m`, `send_packets.m`, `receive_packets.m`, `serial_write_master_xbee.m`, and `serial_read_slave_xbee.m`.

Listing A.5: Programs for the Arduino[5]

```
%% Function setup_arduino
function [ ard_serial_port, Error ] = setup_arduino( device_port )

Error = false;
ard_serial_port = '';

PROMPT01 = ['Please enter COM port # (1 for COM1, etc.) for Arduino'];
TITLE01 = 'COM port';
% Input the serial port Number for the Arduino
ComNum = char(inputdlg(PROMPT01, TITLE01, 1, {num2str(device_port)}));

if isstrprop(ComNum, 'digit')
    ard_serial_port = instrfind('Type', 'serial', 'Port', strcat('COM',
       ComNum), 'Tag', '');
    if isempty(ard_serial_port)
        ard_serial_port = serial(strcat('COM', ComNum));
    else
        fclose(ard_serial_port);
        ard_serial_port = ard_serial_port(1);
    end

    % Set serial port parameters
    ard_serial_port.InputBufferSize = 512;
    ard_serial_port.OutputBufferSize = 512;
    ard_serial_port.BaudRate = 9600;
    ard_serial_port.DataBits = 8;
    ard_serial_port.StopBit = 1;
    ard_serial_port.Timeout = 5;
    ard_serial_port.Parity = 'none';
    ard_serial_port.BytesAvailableFcnMode = 'terminator';
    ard_serial_port.Terminator = 'CR/LF';
    try
        fopen(ard_serial_port);
    catch exception
        fprintf(['\n' exception.message '\n']);
        Error = true;
    end
else
    fprintf('\nInvalid COM port  for Arduino selected\n');
    Error = true;
end


%% Function serial_write_master_arduino
function [control_command] = serial_write_master_arduino(ard_serial_port,
   delta_master, speed_ref)
% This function sends the control commands to the Arduino board.
control_command = sprintf('%f,%f$', delta_master, speed_ref)
fprintf(ard_serial_port, control_command);
end


%% Function serial_write_slave_arduino
function [control_command] = serial_write_slave_arduino(ard_serial_port,
   delta_slave, speed_slave)
```

```
% This function sends the control commands to the Arduino board.
control_command = sprintf('%f,%f$', delta_slave, speed_slave)
fprintf(ard_serial_port, control_command);
end
```

[5]Programs for the Arduino include three separate files: `setup_arduino.m`, `serial_write_master_arduino.m`, and `serial_write_slave_arduino.m`.

Listing A.6: Utility functions[6]

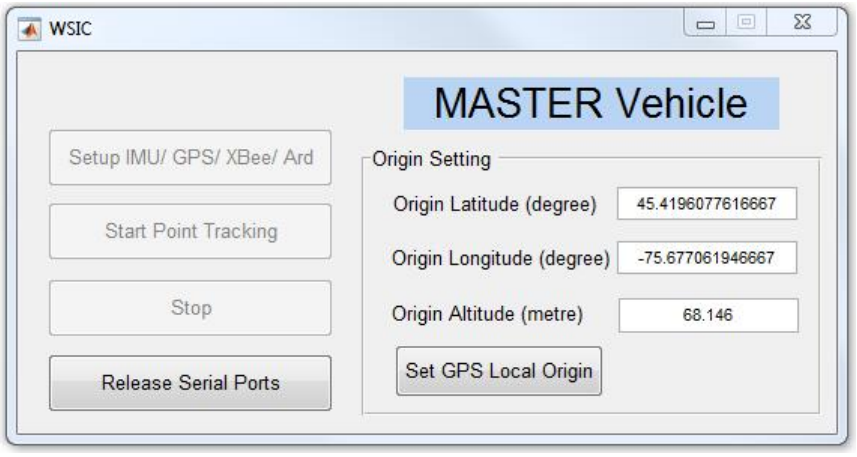
```
%% Function load_folder_subfolder_libraries
function load_folder_subfolder_libraries
% Use genpath in conjunction with addpath to add the current folder
% and its subfolders to the search path
currentFolder = pwd;
p = genpath(currentFolder);
addpath(p);


%% Function distance_point_to_line
function [distance, path_angle ] = distance_point_to_line(point, path)
% Calculates the perpendicular distance of a point to a straight path
% and the orientation of the path with respect to the x-axis
%
% In this function, A, B, and C are the three parameters of a straight
% line in the form of Ax+By+C = 0.
% The following example illustrates how this function can be called:
%
%       point = [x_local, y_local];
%       path = [A, B, C];
%       [distance, path_angle] = distance_point_to_line(point, path)
  x = point(1);
  y = point(2);

  A = path(1);
  B = path(2);
  C = path(3);

  distance = (A * x + B * y + C) / sqrt(A^2 + B^2);
  path_angle = atan2d(-A, B);
end


%% Function offset_distance
function e_offset = offset_distance(point_1, point_0, path)
% Calculates the offset distance between the master and slave
% In this function, A, B, and C are the three parameters of a straight
% line in the form of Ax+By+C = 0.
% The following example illustrates how this function can be called:
%
%       point_1 = [x_local_master, y_local_master];
%       point_0 = [x_local_slave, y_local_slave];
%       path = [A, B, C];
%       e_offset = offset_distance(point_1, point_0, path)
  x_master = point_1(1);
  y_master = point_1(2);

  x_slave = point_0(1);
  y_slave = point_0(2);

  A = path(1);
  B = path(2);
  C = path(3);

  A_p = -1/A;
```

```
  B_p = B;
  C_p = 0;

  distance_master = (A_p * x_master + B_p * y_master + C_p) / sqrt(A_p^2 +
     B_p^2);
  distance_slave = (A_p * x_slave + B_p * y_slave + C_p) / sqrt(A_p^2 + B_p
     ^2);

  e_offset = distance_master - distance_slave;

end


%% Function getCurrentClockStr
function [current_time] = getCurrentClockStr()
% This function returns the current time in string format.
current_time = datestr(clock, 'yyyymmddHHMMSSFFF');
end
```

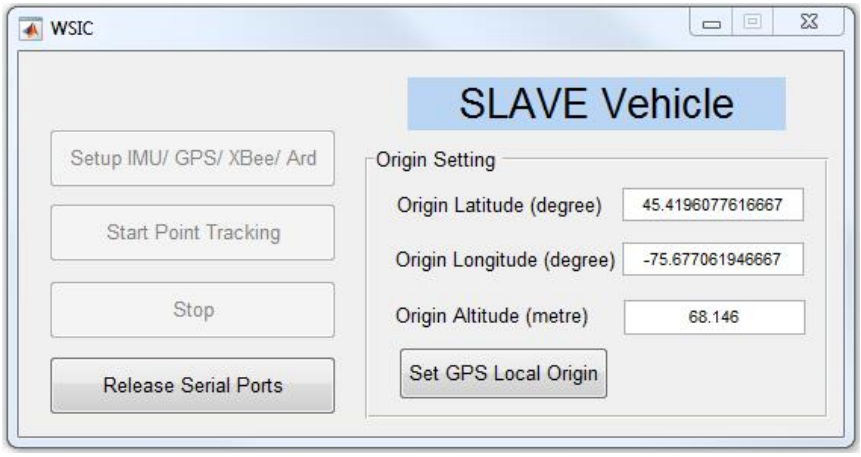
[6]Utility functions used for the program include the following separate files: `load_folder_subfolder_libraries.m`, `distance_point_to_line.m`, `offset_distance.m`, and `getCurrentClockStr.m`.

A.2 MATLAB programs for the stationary mode

The designed GUI for the stationary mode is shown in Figure A.2. After the origin is set, the button for setting up the serial communications for the IMU, GPS, XBee, and Arduino is enabled. To run the sequential point-to-point tracking test, the slave robot needs to enter the point tracking mode first. Then the slave robot waits for the start signal that is to be sent by the master robot. When the master program starts and a start signal is initiated, both robots start moving forward to perform a point tracking process. After both robots reach the target point and stop, they wait for a constant time period, which is to simulate the implement operating process. After the pause is finished, both robots start moving forward and the whole process is repeated automatically. When one test is finished, by clicking the Stop button, the experimental data will be save to the laptop hard drive as Excel files (.xlsx format).



(a)



(b)

Figure A.2: GUI for the stationary operating mode, (a) Master; (b) Slave.

Listing A.7: Programs for the WSIC Master of the stationary mode[7]

```
%% Function WSIC
function varargout = WSIC(varargin)

% WSIC M-file for WSIC.fig
%      WSIC, by itself, creates a new WSIC or raises the existing
%      singleton*.
%
%      H = WSIC returns the handle to a new WSIC or the handle to
%      the existing singleton*.
%
%      WSIC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in WSIC.M with the given input arguments.
%
%      WSIC('Property','Value',...) creates a new WSIC or raises the
%      existing singleton*.  Starting from the left, property value pairs
   are
%      applied to the GUI before WSIC_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
   application
%      stop.  All inputs are passed to WSIC_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WSIC

% Last Modified by GUIDE v2.5 18-Nov-2016 11:32:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @WSIC_OpeningFcn, ...
                   'gui_OutputFcn',  @WSIC_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before WSIC is made visible.
function WSIC_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to WSIC (see VARARGIN)

% Choose default command line output for WSIC
handles.output = hObject;

% Set control buttons Enable states
set(handles.btn_setup_serial,'Enable','off');
set(handles.btn_start_tracking,'Enable','off');
set(handles.btn_stop_tracking,'Enable','off');

% Update handles structure
guidata(hObject, handles);

% *** load all libraries
load_folder_subfolder_libraries;
% UIWAIT makes WSIC wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = WSIC_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btn_setup_serial.
function btn_setup_serial_Callback(hObject, eventdata, handles)
% hObject    handle to btn_setup_serial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IMU_SERIAL_PORT;
global IMU_Error;

global GPS_SERIAL_PORT;
global GPS_Error;

global XBEE_SERIAL_PORT;
global XBEE_Error;

global ARD_SERIAL_PORT;
global ARD_Error;

% Start setting up serial communications
% COM port numbers depend on physical connections, check in Device Manager
[IMU_SERIAL_PORT, IMU_Error] = setup_imu(6);
[GPS_SERIAL_PORT, GPS_Error] = setup_gps(38);
[XBEE_SERIAL_PORT, XBEE_Error] = setup_xbee(34);
[ARD_SERIAL_PORT, ARD_Error] = setup_arduino(33);

if (~GPS_Error && ~IMU_Error && ~XBEE_Error && ~ARD_Error)
    % Enable "Start" and "Stop" buttons
```

```
    set(handles.btn_start_tracking,'Enable','on');
    set(handles.btn_stop_tracking,'Enable','on');
end
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in btn_start_tracking.
function btn_start_tracking_Callback(hObject, eventdata, handles)
% hObject    handle to btn_start_tracking (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_RUNNING;
IS_RUNNING = true;

global IMU_SERIAL_PORT;
global IMU_DATA;

global GPS_SERIAL_PORT;
global GPS_ORIGIN;
global GPS_DATA;

global XBEE_SERIAL_PORT;
global CONTROL_DATA;

global ARD_SERIAL_PORT;

set(handles.btn_setup_serial,'Enable','off');
set(handles.btn_stop_tracking,'Enable','on');

%%% Main control loop
for index = 1:5
    slave_status = 0; % Reset vehicle status to Stop (=0)
    master_status = 0;
    % Start GPS data reading and saving
    [gps_first_packet, ~] = read_gps(GPS_SERIAL_PORT, GPS_ORIGIN);
    GPS_DATA = [fieldnames(gps_first_packet)'; struct2cell(gps_first_packet
       )'];
    % Start IMU data reading and saving
    [imu_first_packet, ~] = read_imu_EulerAngles(IMU_SERIAL_PORT);
    IMU_DATA = [fieldnames(imu_first_packet)'; struct2cell(imu_first_packet
       )'];
    % Start control values sending and saving
    [control_first_packet, ~, ~] = master_point_track(index,
       ARD_SERIAL_PORT, gps_first_packet, gps_first_packet,
       imu_first_packet);
    CONTROL_DATA = [fieldnames(control_first_packet)'; struct2cell(
       control_first_packet)'];
    % Send index # to the Slave
    master_xbee_serial_write(XBEE_SERIAL_PORT, index);

    % Start Master vehicle point tracking
    while 1
        if master_status == 0
            [gps_packet, gps_read_error] = read_gps(GPS_SERIAL_PORT,
               GPS_ORIGIN);
```

```
        if (~gps_read_error)
            GPS_DATA = [GPS_DATA; struct2cell(gps_packet)'];
        end
        pause(0.00001);

        [imu_packet, imu_read_error] = read_imu_EulerAngles(
           IMU_SERIAL_PORT);
        if (~imu_read_error)
            IMU_DATA = [IMU_DATA; struct2cell(imu_packet)'];
        end
        pause(0.00001);

        if (~gps_read_error && ~imu_read_error)
            [control_packet, control_error, master_stop_flag] =
               master_point_track(index, ARD_SERIAL_PORT,
               gps_first_packet, gps_packet, imu_packet);
            master_status = master_stop_flag;
            if (~control_error)
                CONTROL_DATA = [CONTROL_DATA; struct2cell(
                   control_packet)'];
            end
            pause(0.00001);
        end
    end

    % Keep checking the status of the Slave from XBee
    if slave_status == 0 % if Slave not stopped, keep reading the XBee
        port
        [slave_stop_flag] = master_xbee_serial_read(XBEE_SERIAL_PORT);
        slave_status = slave_stop_flag;
    end
    pause(0.00001);

    % If both the Master and Slave stopped, point tracking is finished.
    if (master_status == true && slave_status == 1)
       pause(0.1);
       break
    end
end

% Save data to Excel files
s1=sprintf('GPS_data_Master_%d.xlsx',index);
s2 = 'G:\';
filename1 = strcat(s2, s1);
xlswrite(filename1, GPS_DATA);
s3=sprintf('IMU_data_Master_%d.xlsx',index);
s4 = 'G:\';
filename2 = strcat(s4, s3);
xlswrite(filename2, IMU_DATA);
s5=sprintf('Control_data_Master_%d.xlsx',index);
s6 = 'G:\';
filename3 = strcat(s6, s5);
xlswrite(filename3, CONTROL_DATA);

% This pause is to simulate the implement operating process.
pause(12);
```

```
end

% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in btn_stop_tracking.
function btn_stop_tracking_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_stop_gps (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_RUNNING;
IS_RUNNING = false;

set(handles.btn_setup_serial,'Enable','off');
set(handles.btn_start_tracking,'Enable','on');
set(handles.btn_stop_tracking,'Enable','off');


% --- Executes on button press in btn_release_ports.
function btn_release_ports_Callback(hObject, eventdata, handles)
% hObject    handle to btn_release_ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(instrfindall);
set(handles.btn_setup_serial,'Enable','on');
set(handles.btn_start_tracking,'Enable','off');
set(handles.btn_stop_tracking,'Enable','off');


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

isRunning = getappdata(handles.figure1, 'isRunning');
setappdata(handles.figure1, 'isRunning', false);
clear global;
pause(1);
% Hint: delete(hObject) closes the figure
delete(hObject);


function edit_GpsLat_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsLat_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsLat_Ori as text
%        str2double(get(hObject,'String')) returns contents of
   edit_GpsLat_Ori as a double
global ORI_LATITUDE;

ori_latitude = str2double(get(hObject,'String'));
if isnan(ori_latitude)
```

```
    h = msgbox('The Latitude should be a double value in degrees!');
else
    if (ori_latitude >= 0 && ori_latitude <= 90)
        ORI_LATITUDE = ori_latitude;
    else
        h = msgbox(['The Latitude should be between [' num2str(0) ',' ...
                num2str(90) '] degrees!']);
    end
end
set(hObject,'String',sprintf('%0.7f', ORI_LATITUDE));


% --- Executes during object creation, after setting all properties.
function edit_GpsLat_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsLat_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','45.4196077616667'); % default latitude value for the
   origin


function edit_GpsLon_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsLon_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsLon_Ori as text
%        str2double(get(hObject,'String')) returns contents of
   edit_GpsLon_Ori as a double
global ORI_LONGITUDE;

ori_longitude = str2double(get(hObject,'String'));
if isnan(ori_longitude)
    h = msgbox('The Longitude should be a double value in degrees!');
else
    if (ori_longitude >= -180 && ori_longitude <= 180)
        ORI_LONGITUDE = ori_longitude;
    else
        h = msgbox(['The Longitude should be between (' num2str(-180) ','
           ...
                num2str(180) '] degrees!']);
    end
end
set(hObject,'String',sprintf('%0.7f', ORI_LONGITUDE));

% --- Executes during object creation, after setting all properties.
function edit_GpsLon_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsLon_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','-75.677061946667'); % default longitude value for the
    origin


function edit_GpsAlt_Ori_Callback(hObject, eventdata, handles)
% hObject    handle to edit_GpsAlt_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_GpsAlt_Ori as text
%        str2double(get(hObject,'String')) returns contents of
   edit_GpsAlt_Ori as a double
global ORI_ALTITUDE;

ori_altitude = str2double(get(hObject,'String'));
if ~isnan(ori_altitude)
    h = msgbox('The Altitude should be a double value in meters!');
else
    ORI_ALTITUDE = ori_altitude;
end
% Uses up to 3 floating points to show values in milimeters precision
set(hObject,'String', sprintf('%0.3f', ORI_ALTITUDE));

% --- Executes during object creation, after setting all properties.
function edit_GpsAlt_Ori_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_GpsAlt_Ori (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'String','68.146'); % default altitude value for the origin


% --- Executes on button press in btn_set_origin.
function btn_set_origin_Callback(hObject, eventdata, handles)
% hObject    handle to btn_set_origin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ORI_LATITUDE;
global ORI_LONGITUDE;
global ORI_ALTITUDE;

global GPS_ORIGIN;

if get(hObject, 'value')
```

```
    % Read origin setting values from text boxes
    ORI_LATITUDE = str2double(get(handles.edit_GpsLat_Ori, 'String'));
    ORI_LONGITUDE = str2double(get(handles.edit_GpsLon_Ori, 'String'));
    ORI_ALTITUDE = str2double(get(handles.edit_GpsAlt_Ori, 'String'));

    GPS_ORIGIN.altitude = ORI_ALTITUDE;
    GPS_ORIGIN.longitude = ORI_LONGITUDE;
    GPS_ORIGIN.latitude = ORI_LATITUDE;
    % When the origin is set, enable the "Start" button.
    set(handles.btn_setup_serial,'Enable','on');
end


%% Function master_point_track
function [control_packet, error, stop_flag] = master_point_track(index,
   ard_serial_port, gps_first_packet, gps_packet, imu_packet)
% master_point_track.m is a function written for the Master to perform
% one forward point tracking process. The input index determines the
% coordinates of the target point. The control signals are calculated
% and sent to the Arduino.

%%% Define the desired straight path by A, B and C parameters Ax+By+C = 0
% PATH1 y = 0.5997x + 4.2276
PATH = [-0.5997 1 -4.2276]; % PATH = [A, B, C];
theta_d = 30.951; % desired orientation angle
% Calcuated coordinates of target points
% x_d_group = [0; 1.28641; 2.57282; 3.85923; 5.14564]; % 1.5-m implement
% y_d_group = [4.2276; 4.99906; 5.77052; 6.54198; 7.31344];
% implement_width = 1.5;
% x_d_group = [0; 1.71521; 3.43042; 5.14563; 6.86084]; % 2.0-m implement
% y_d_group = [4.2276; 5.25621; 6.28482; 7.31343; 8.34204];
% implement_width = 2.0;
x_d_group = [0; 2.14402; 4.28804; 6.43206; 8.57608; 10.7201]; % 2.5-m
   implement
y_d_group = [4.2276; 5.51377; 6.79914; 8.08491; 9.37068; 10.65645];
implement_width = 2.5;
% x_d_group = [0; 2.57282; 5.14564; 7.71846; 10.29128]; % 3-m implement
% y_d_group = [4.2276; 5.77052; 7.31344; 8.85636; 10.39928];
% implement_width = 3;

% PATH2 y = 0.5997x - 3.351634
% PATH = [-0.5997 1 3.351634]; % PATH = [A, B, C];
% theta_d = 30.9511; % desired orientation angle
% Calcuated coordinates of target points
% x_d_group = [3.34300; 4.62940; 5.91581; 7.20222; 8.48863]; % 1.5-m
   implement
% y_d_group = [-1.34684; -0.57538; 0.19608; 0.96754; 1.73900];
% implement_width = 1.5;
% x_d_group = [3.34300; 5.05821; 6.77342; 8.48863; 10.20384]; % 2.0-m
   implement
% y_d_group = [-1.34684; -0.31823; 0.71038; 1.73900; 2.76761];
% implement_width = 2.0;
% x_d_group = [3.34300; 5.48701; 7.63103; 9.77505; 11.91907]; % 2.5-m
   implement
% y_d_group = [-1.34684; -0.06107; 1.22470; 2.51047; 3.79624];
% implement_width = 2.5;
```

```
% x_d_group = [3.34300; 5.92127; 8.49409; 11.06691; 13.63973]; % 3-m
    implement
% y_d_group = [-1.34684; 0.19608; 1.74227; 3.28519; 4.82811];
% implement_width = 3;

% Desired start point coordinates
x_d0 = x_d_group(index);
y_d0 = y_d_group(index);
% Desired target point coordinates
x_d = x_d_group(index+1);
y_d = y_d_group(index+1);
% Initial position, true start point of the robot
x_0 = gps_first_packet.x;
y_0 = gps_first_packet.y;
% Initial distance error
rho_0 = sqrt((x_d - x_0)^2 + (y_d - y_0)^2);
d_perpen_0 = distance_to_perpendicular_line([x_0, y_0], [x_d, y_d], PATH);

% Constants used in Velocity and Delta formulas
L_wheelbase = 0.495; % robot wheelbase 495 mm
n = 8;  kv = 0.3; % velocity control gains
k1 = 0.5; ku = 0.6; % steering angle control gains

% Initialize output arguments
error = false;
stop_flag = false;
% Variables for data saving
control_packet.Delta_Command = 0;
control_packet.Speed_Command = 0;
control_packet.Speed_gps = 0;
control_packet.Lateral_error = 0;
control_packet.Offset_error = 0;
control_packet.Orientation_error = 0;
control_packet.Rho_distance = 0;
control_packet.Alpha_angle_v = 0;
control_packet.Beta_angle_v = 0;
control_packet.Date = '';

try
    % Get data from GPS
    x_local = gps_packet.x;
    y_local = gps_packet.y;
    speed_gps = gps_packet.speed;

    % Get yaw angle from the IMU sensor, [-180, 180]
    % North = 0, East = 90, West = -90 degree
    yaw_angle = imu_packet.yaw;
    % Calibrate the orientation angle w.r.t the straight line of slope
    % 30.951 degrees, the true yaw angle should be 59.049 degrees
    yaw_angle = yaw_angle -64.42 + 59.049;
    % Orientation angle of the vehicle with respect to the East
    % East = 0, North = 90, South = -90 degree
    % Below is the corversion from Yaw angle to Orientation angle.
    if (yaw_angle >=-90 && yaw_angle <=180)
        orientation_angle = yaw_angle * (-1) + 90;
    elseif(yaw_angle < -90 && yaw_angle >=-180)
        orientation_angle = yaw_angle * (-1) - 270;
```

```
end

% Calculate the deviation distance and path angle
[deviation, PATH_ANGLE] = distance_point_to_line([x_local, y_local],
    PATH);
% Calculate the orientation error
theta_p = orientation_angle - PATH_ANGLE;

Delta_x = x_d - x_local;
Delta_y = y_d - y_local;
psi_angle = atan2d(Delta_y, Delta_x);
alpha_angle = psi_angle - orientation_angle;
beta_angle = psi_angle - theta_d;

% Calculate the heading and orientation errors based on the virtual
    target (1 m ahead)
Delta_x_virtual = x_d + 1/1.166036059 - x_local;
Delta_y_virtual = y_d + 0.5997/1.166036059 - y_local;
psi_angle_virtual = atan2d(Delta_y_virtual, Delta_x_virtual);
alpha_angle_v = psi_angle_virtual - orientation_angle;
beta_angle_v = psi_angle_virtual - theta_d;
% Calculate the distance error
rho_distance = sqrt(Delta_x^2 + Delta_y^2);
rho_distance_v = sqrt(Delta_x_virtual^2 + Delta_y_virtual^2);

% Distance of the robot to the desired start line
distance_perpendicular = distance_to_perpendicular_line([x_local,
    y_local], [x_d0, y_d0], PATH);
% Distance of the robot to the desired taret line
d_perpen = distance_to_perpendicular_line([x_local, y_local], [x_d, y_d
    ], PATH);
% Calculate the offset error
offset_error = implement_width - distance_perpendicular;

%%% Calculate control commands
% Calculate speed commands
speed_command = (-2^n*(d_perpen - 0.5*d_perpen_0)^n / d_perpen_0^n + 1)
    *kv;
% Give the robot an initial speed to start. This value depends on the
    motor responsiveness.
if (abs(speed_command) <= 0.1)
    speed_command = 0.1;
end

% When the robot reaches a distance of the implement width, stop.
if(distance_perpendicular >= implement_width-0.02) % a 0.02-m margin to
     prevent overshoot
    speed_command = 0;
end

% Calculate steering commands
c_command = sind(beta_angle_v)*cosd(beta_angle_v)/(rho_distance_v*cosd(
    alpha_angle_v)) + sind(alpha_angle_v)/rho_distance_v+...
    2*k1*cosd(2*beta_angle_v)*tand(alpha_angle_v)/rho_distance_v + ku*
        tand(alpha_angle_v) + k1*ku*sind(2*beta_angle_v)/cosd(
        alpha_angle_v);
```

```
    delta_command = atand(L_wheelbase * c_command); % delta = atan(L * c)

    % Restrict the steering angle to saturation range [-26.98, 26.98]
    if (delta_command > 26.98)
        delta_command = 26.98;
    elseif (delta_command < -26.98)
        delta_command = -26.98;
    end

    % The following conversion is needed since delta_command is the
    % steering angle for the virtual wheel in the middle of the axle,
    % while the true control angle is on the left wheel.
    if (delta_command >= 0)
        delta_command_left = atand( 0.495*tand(delta_command) /
            (0.495-0.18*tand(delta_command)) );
    elseif (delta_command < 0)
        delta_command_left = atand( 0.495*tand(delta_command) /
            (0.495+0.18*tand(delta_command)) );
    end

    % Send control commands to the robot through Arduino serial port
    serial_write_arduino(ard_serial_port, delta_command_left, speed_command
        );

    % Save data variables
    control_packet.Delta_Command = delta_command;
    control_packet.Speed_Command = speed_command;
    control_packet.Speed_gps = speed_gps;
    control_packet.Lateral_error = deviation;
    control_packet.Offset_error = offset_error;

    control_packet.Orientation_error = theta_p;
    control_packet.Rho_distance = rho_distance;
    control_packet.Alpha_angle_v = alpha_angle_v;
    control_packet.Beta_angle_v = beta_angle_v;
    control_packet.Date = getCurrentClockStr;

    if (distance_perpendicular >= implement_width - 0.10 && speed_command
        == 0)
            pause(0.1);
            stop_flag = true; % vehicle stopped, point tracking finished
            return
    end

catch exception
    fprintf(exception.message);
    error = true;
end


%% Function master_xbee_serial_write
function [final_command] = master_xbee_serial_write(xbee_serial_port, index
    )
% To send the index number to the Slave over the XBee serial port
    final_command = sprintf('%f', index)
    fprintf(xbee_serial_port, final_command);
end
```

```
%% Function master_xbee_serial_read
function [slave_status] = master_xbee_serial_read(xbee_serial_port)
% To read Slave status from the XBee serial port
try
    serial_data = fgets(xbee_serial_port);
    slave_status = sscanf(serial_data, '%d');
catch exception
    fprintf(exception.message);
end
end
```

```
%% Function distance_to_perpendicular_line
function distance = distance_to_perpendicular_line(point_1, point_0, path)
% Calculates the distance from the robot to a line which passes through
% the start point and is perpendicular to the desired path
%
%   In this function, A, B, and C are the three parameters of the disred
%   straight path, in the form of Ax+By+C = 0.
%
%   The following example illustrates how this function can be called:
%
%       point_1 = [x_local, y_local];
%       point_0 = [x_d0, y_d0];
%       path = [A, B, C];
%       perpendicular_distance = distance_to_perpendicular_line(point_1,
   point_0, path)
  x1 = point_1(1);
  y1 = point_1(2);

  x0 = point_0(1);
  y0 = point_0(2);

  A = path(1);
  B = path(2);
  C = path(3);

  A_p = -1/A;
  B_p = B;
  C_p = -A_p*x0 - B_p * y0;

  distance = (A_p * x1 + B_p * y1 + C_p) / sqrt(A_p^2 + B_p^2);
end
```

[7]Programs used for the Master under the stationary mode include five separate files: `WSIC.m`, `master_point_track.m`, `master_xbee_serial_write.m`, `master_xbee_serial_read.m`, and `distance_to_perpendicular_line.m`. The other files including the GPS, IMU, XBee, and Arduino setup functions, GPS and IMU reading functions, Arduino writing functions, and utility functions are the same as the ones programmed for the mobile mode.

Listing A.8: Programs for the WSIC Slave of the stationary mode[8]

```
% ---- Executes on button press in btn_start_tracking.
function btn_start_tracking_Callback(hObject, eventdata, handles)
% hObject    handle to btn_start_tracking (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global IS_RUNNING;
IS_RUNNING = true;

global IMU_SERIAL_PORT;
global IMU_DATA;

global GPS_SERIAL_PORT;
global GPS_ORIGIN;
global GPS_DATA;

global XBEE_SERIAL_PORT;
global CONTROL_DATA;

global ARD_SERIAL_PORT;

set(handles.btn_setup_serial,'Enable','off');
set(handles.btn_stop_tracking,'Enable','on');

while (IS_RUNNING)
    % Read the index number from the XBee serial port
    [index] = slave_xbee_serial_read(XBEE_SERIAL_PORT);
    if isempty(index)
        % The pause is needed to not block the GUI!
        pause(0.00001);
        continue; % keep reading until an index number is received
    end

    % Start GPS data reading and saving
    [gps_first_packet, ~] = read_gps(GPS_SERIAL_PORT, GPS_ORIGIN);
    GPS_DATA = [fieldnames(gps_first_packet)'; struct2cell(gps_first_packet
       )'];
    % Start IMU data reading and saving
    [imu_first_packet, ~] = read_imu_EulerAngles(IMU_SERIAL_PORT);
    IMU_DATA = [fieldnames(imu_first_packet)'; struct2cell(imu_first_packet
       )'];
    % Start control values sending and saving
    [control_first_packet, ~, ~] = slave_point_track(index, ARD_SERIAL_PORT
       , gps_first_packet, gps_first_packet, imu_first_packet);
    CONTROL_DATA = [fieldnames(control_first_packet)'; struct2cell(
       control_first_packet)'];

    % Start Slave vehicle point tracking
    while 1
        [gps_packet, gps_read_error] = read_gps(GPS_SERIAL_PORT, GPS_ORIGIN
           );
        if (~gps_read_error)
            if (~isempty(gps_packet))
                GPS_DATA = [GPS_DATA; struct2cell(gps_packet)'];
            end
        end
```

```
        pause(0.00001);

        [imu_packet, imu_read_error] = read_imu_EulerAngles(IMU_SERIAL_PORT
            );
        if (~imu_read_error)
            IMU_DATA = [IMU_DATA; struct2cell(imu_packet)'];
        end
        pause(0.00001);

        if (~gps_read_error && ~imu_read_error)
            [control_packet, control_error, stop_flag] = slave_point_track(
                index, ARD_SERIAL_PORT, gps_first_packet, gps_packet,
                imu_packet);
            if (~control_error)
                CONTROL_DATA = [CONTROL_DATA; struct2cell(control_packet)
                    '];
            end
            pause(0.00001);
        end

        if stop_flag == false % if not stopped, keep sending value '0'
            slave_xbee_serial_write(XBEE_SERIAL_PORT, 0);
        end

        if stop_flag == true % if stopped, send value '1'
            slave_xbee_serial_write(XBEE_SERIAL_PORT, 1);
            break
        end
    end

    % Save data to Excel files
    s1=sprintf('GPS_data_Slave_%d.xlsx',index);
    s2 = 'G:\';
    filename1 = strcat(s2, s1);
    xlswrite(filename1, GPS_DATA);
    s3=sprintf('IMU_data_Slave_%d.xlsx',index);
    s4 = 'G:\';
    filename2 = strcat(s4, s3);
    xlswrite(filename2, IMU_DATA);
    s5=sprintf('Control_data_Slave_%d.xlsx',index);
    s6 = 'G:\';
    filename3 = strcat(s6, s5);
    xlswrite(filename3, CONTROL_DATA);
end

% Update handles structure
guidata(hObject, handles);


%% Function slave_point_track
function [control_packet, error, stop_flag] = slave_point_track(index,
    ard_serial_port, gps_first_packet, gps_packet, imu_packet)
% slave_point_track.m is a function written for the Slave to perform
% one forward point tracking process. The input index determines the
% coordinates of the target point. The control signals are calculated
% and sent to the Arduino.
```

```
%%% Define the desired straight path by A, B and C parameters Ax+By+C = 0
% PATH1 y = 0.5997x + 4.2276
% PATH = [-0.5997 1 -4.2276]; % PATH = [A, B, C];
% theta_d = 30.9511; % desired orientation angle
% Calcuated coordinates of target points
% x_d_group = [0; 1.2864096; 2.57282; 3.85923; 5.14564]; % 1.5-m implement
% y_d_group = [4.2276; 4.99906; 5.77052; 6.54198; 7.31344];
% implement_width = 1.5;
% x_d_group = [0; 1.71521; 3.43042; 5.14563; 6.86084]; % 2.0-m implement
% y_d_group = [4.2276; 5.25621; 6.28482; 7.31343; 8.34204];
% implement_width = 2.0;
% x_d_group = [0; 2.14402; 4.28804; 6.43206; 8.57608]; % 2.5-m implement
% y_d_group = [4.2276; 5.51377; 6.79914; 8.08491; 9.37068];
% implement_width = 2.5;
% x_d_group = [0; 2.57282; 5.14564; 7.71846; 10.29128]; % 3-m implement
% y_d_group = [4.2276; 5.77052; 7.31344; 8.85636; 10.39928];
% implement_width = 3;

% PATH2 y = 0.5997x - 3.351634
PATH = [-0.5997 1 3.351634]; % PATH = [A, B, C];
theta_d = 30.951; % desired orientation angle
% Calcuated coordinates of target points
% x_d_group = [3.34300; 4.62940; 5.91581; 7.20222; 8.48863]; % 1.5-m
   implement
% y_d_group = [-1.34684; -0.57538; 0.19608; 0.96754; 1.73900];
% implement_width = 1.5;
% x_d_group = [3.34300; 5.05821; 6.77342; 8.48863; 10.20384]; % 2.0-m
   implement
% y_d_group = [-1.34684; -0.31823; 0.71038; 1.73900; 2.76761];
% implement_width = 2.0;
x_d_group = [3.34300; 5.48701; 7.63103; 9.77505; 11.91907; 14.06309]; %
   2.5-m implement
y_d_group = [-1.34684; -0.06107; 1.22470; 2.51047; 3.79624; 5.08201];
implement_width = 2.5;
% x_d_group = [3.3429927; 5.9212727; 8.4940927; 11.0669127; 13.6397327]; %
   3-m implement
% y_d_group = [-1.3468413; 0.1960784; 1.7422734; 3.2851935; 4.8281137];
% implement_width = 3;

% Desired start point coordinates
x_d0 = x_d_group(index);
y_d0 = y_d_group(index);
% Desired target point coordinates
x_d = x_d_group(index+1);
y_d = y_d_group(index+1);
% Initial position, true start point of the robot
x_0 = gps_first_packet.x;
y_0 = gps_first_packet.y;
% Initial distance error
rho_0 = sqrt((x_d - x_0)^2 + (y_d - y_0)^2);
d_perpen_0 = distance_to_perpendicular_line([x_0, y_0], [x_d, y_d], PATH);

% Constants used in Velocity and Delta formula
L_wheelbase = 0.495; % robot wheelbase 495 mm
n = 8;  kv = 0.3; % velocity control gains
k1 = 0.5; ku = 0.6; % steering angle control gains
```

```
% Initialize output arguments
error = false;
stop_flag = false;
% Variables for data saving
control_packet.Delta_Command = 0;
control_packet.Speed_Command = 0;
control_packet.Speed_gps = 0;
control_packet.Lateral_error = 0;
control_packet.Offset_error = 0;
control_packet.Orientation_error = 0;
control_packet.Rho_distance = 0;
control_packet.Alpha_angle_v = 0;
control_packet.Beta_angle_v = 0;
control_packet.Date = '';

try
    % Get data from GPS
    x_local = gps_packet.x;
    y_local = gps_packet.y;
    speed_gps = gps_packet.speed;

    % Get yaw angle from the IMU sensor, [-180, 180]
    % North = 0, East = 90, West = -90 degree
    yaw_angle = imu_packet.yaw;
    % Calibrate the orientation angle w.r.t the straight line of slope
    % 30.951 degrees, the true yaw angle should be 59.049 degrees
    yaw_angle = yaw_angle  + 2.9 + 59.049;
    % Orientation angle of the vehicle with respect to the East
    % East = 0, North = 90, South = -90 degree
    % Below is the corversion from Yaw angle to Orientation angle.
    if (yaw_angle >=-90 && yaw_angle <=180)
        orientation_angle = yaw_angle * (-1) + 90;
    elseif(yaw_angle < -90 && yaw_angle > -180)
        orientation_angle = yaw_angle * (-1) - 270;
    end

    % Calculate the deviation distance and path angle
    [deviation, PATH_ANGLE] = distance_point_to_line([x_local, y_local],
       PATH);
    % Calculate the orientation error
    theta_p = orientation_angle - PATH_ANGLE;

    Delta_x = x_d - x_local;
    Delta_y = y_d - y_local;
    psi_angle = atan2d(Delta_y, Delta_x);
    alpha_angle = psi_angle - orientation_angle;
    beta_angle = psi_angle - theta_d;

    % Calculate the heading and orientation errors based on the virtual
       target (1 m ahead)
    Delta_x_virtual = x_d + 1.0/1.166036059 - x_local;
    Delta_y_virtual = y_d + 1.0*0.5997/1.166036059 - y_local;
    psi_angle_virtual = atan2d(Delta_y_virtual, Delta_x_virtual);
    alpha_angle_v = psi_angle_virtual - orientation_angle;
    beta_angle_v = psi_angle_virtual - theta_d;

    % Calculate distance error
```

```
rho_distance = sqrt(Delta_x^2 + Delta_y^2);
rho_distance_v = sqrt(Delta_x_virtual^2 + Delta_y_virtual^2);

% Distance of the robot to the desired start line
distance_perpendicular = distance_to_perpendicular_line([x_local,
    y_local], [x_d0, y_d0], PATH);
% Distance of the robot to the desired taret line
d_perpen = distance_to_perpendicular_line([x_local, y_local], [x_d, y_d
    ], PATH);
     % Calculate the offset error
     offset_error = implement_width - distance_perpendicular;

%%% Calculate control commands
speed_command = (-2^n*(d_perpen - 0.5*d_perpen_0)^n / d_perpen_0^n + 1)
    *kv;
% Give the robot an initial speed to start. This value depends on the
    motor responsiveness.
if (abs(speed_command) <= 0.10)
     speed_command = 0.10;
end

% When the robot reaches a distance of the implement width, stop.
if(distance_perpendicular >= implement_width)
     speed_command = 0;
end

% Calculate steering commands
c_command = sind(beta_angle_v)*cosd(beta_angle_v)/(rho_distance_v*cosd(
    alpha_angle_v)) + sind(alpha_angle_v)/rho_distance_v+...
     2*k1*cosd(2*beta_angle_v)*tand(alpha_angle_v)/rho_distance_v + ku*
         tand(alpha_angle_v) + k1*ku*sind(2*beta_angle_v)/cosd(
         alpha_angle_v);

delta_command = atand(L_wheelbase * c_command); % delta = atan(L * c)

% Restrict the steering angle to saturation range [-26.98, 26.98]
if (delta_command > 26.98)
     delta_command = 26.98;
elseif (delta_command < -26.98)
     delta_command = -26.98;
end

% The following conversion is needed since delta_command is the
% steering angle for the virtual wheel in the middle of the axle,
% while the true control angle is on the left wheel.
if (delta_command >= 0)
     delta_command_left = atand( 0.495*tand(delta_command) /
         (0.495-0.18*tand(delta_command)) );
elseif (delta_command < 0)
     delta_command_left = atand( 0.495*tand(delta_command) /
         (0.495+0.18*tand(delta_command)) );
end

% Send control commands to the robot through Arduino serial port
serial_write_arduino(ard_serial_port, delta_command_left, speed_command
    );
```

```
    % Save data variables
    control_packet.Delta_Command = delta_command;
    control_packet.Speed_Command = speed_command;
    control_packet.Speed_gps = speed_gps;
    control_packet.Lateral_error = deviation;
    control_packet.Offset_error = offset_error;
    control_packet.Orientation_error = theta_p;
    control_packet.Rho_distance = rho_distance;

    control_packet.Alpha_angle_v = alpha_angle_v;
    control_packet.Beta_angle_v = beta_angle_v;
    control_packet.Date = getCurrentClockStr;

    if (distance_perpendicular >= implement_width-0.10 && speed_command ==
       0)
        pause(0.1);
        stop_flag = true; % vehicle stopped, point tracking finished
        return
    end

catch exception
    fprintf(exception.message);
    error = true;
end


%% Function slave_xbee_serial_read
function [index] = slave_xbee_serial_read(serial_port)
% To read the index number from the XBee serial port
try
    serial_data = fgetl(serial_port);
    index = sscanf(serial_data, '%d')
catch exception
    fprintf(exception.message);
end
end


%% Function slave_xbee_serial_write
function slave_xbee_serial_write(xbee_serial_port, value_sent)
% To send Slave status to the Master over the XBee serial port
    slave_status = sprintf('%d', value_sent)
    fprintf(xbee_serial_port, slave_status);
end
```

[8]Programs used for the Slave under the stationary mode include five separate files: `WSIC.m` (only the Start button Callback function is listed), `slave_point_track.m`, `slave_xbee_serial_write.m`, `slave_xbee_serial_read.m`, and `distance_to_perpendicular_line.m`. The other files including the GPS, IMU, XBee, and Arduino setup functions, GPS and IMU reading functions, Arduino writing functions, and utility functions are the same as the ones programmed for the mobile mode.

A.3 Arduino program for robot motor control

Listing A.9: Arduino program for robot motor control

```
/*  This program is written for the Arduino board to receive speed
 *  and steering commands from the upper-level host and control
 *  the motors accordingly through the Sabertooth motor driver.
 *  The Sabertooth libraries are provided by the company Dimension
 *  Engineering. The PID library is downloaded from the GitHub
 *  platform (link: https://github.com/br3ttb/Arduino-PID-Library).
*/

// ******************************************************
// Libraries
// ******************************************************
#include <SabertoothSimplified.h>  // Sabertooth motor driver
#include <Sabertooth.h>
#include <SoftwareSerial.h>  // Serial Library
#include <Arduino.h>  // Arduino
#include <avr/io.h>  // General definations of registers
#include <util/delay.h>  // Delay functions
#include <PID_v1.h>  // PID Library

// ******************************************************
// Hardware Pin Definitions
// ******************************************************
#define SAB_RX 25 // not used
#define SAB_PORT 43 // Sabertooth S1
#define SAB_ESTOP 45 // Sabertooth S2

#define SteerPot A0 // Potentiomenter measuring the steering of the front
    wheels

// ******************************************************
// Define variables and constants
// ******************************************************
int DriveMotorValue = 0; // DriveMotorValue -127~127

float delta_req = 0; // requested steering command
float velocity_req = 0; // requested velocity command

double potRest, angle_act, SteerMotorValue, angle_req;
// Define tuning parameters for the steering PID
double aggKp=1.9, aggKi=0.3, aggKd=0.1; // agressive parameters, not used
double consKp=1.5, consKi=0.25, consKd=0.05; // conservative parameters

// Initialize software serial for the motor controller
SoftwareSerial SWSerial(SAB_RX, SAB_PORT);
SabertoothSimplified DriveMotor(SWSerial);

// Initialize steering motor PID control
PID myPIDSteer(&angle_act, &SteerMotorValue, &angle_req, consKp, consKi,
    consKd, REVERSE);

void setup()
{
  // Initialize serial ports
```

```
  Serial.begin(38400); // for saving the steering angle values
  Serial3.begin(9600); // for receiving control commands
  SWSerial.begin(9600); // for controlling the Sabertooth motor driver
  Serial.flush(); // clear

  pinMode (SAB_ESTOP, OUTPUT);
  allStop(); // stop all motors

  // Read the input commands from Serial3
  if(Serial3.available() > 0)
  {
    delta_req = Serial3.parseFloat();
    velocity_req = Serial3.parseFloat();

    if (Serial3.read() == '$') //done transmitting
    {
      angle_req = delta_req;
      angle_req = constrain (angle_req,-32, 32);
      velocity_req = constrain (velocity_req,-0.8, 0.8);
      if (velocity_req >=0)
      {DriveMotorValue = -(velocity_req*72.504+4.254);} // calibration from
          velocity commands to motor control values
      else
      {DriveMotorValue = -(velocity_req*72.701-3.8011);}
    }
  }
  potRest = (double)analogRead(SteerPot);
  angle_act = angleCali(potRest); // calibration from steering pot readings
      to steering angles
  angle_req = 0;
  double gap = abs(angle_req - angle_act);
  if(abs(gap) < 3 || abs(gap) > 50 ) // if steering error < 3 or > 50, the
     steering motor does not response
  {
    myPIDSteer.SetTunings(0, 0, 0);
  }
   else
  {
   myPIDSteer.SetTunings(consKp, consKi, consKd);
  }

  myPIDSteer.SetMode(AUTOMATIC);
  DriveMotor.motor(2, SteerMotorValue);
  DriveMotor.motor(1, DriveMotorValue);
}

void loop()
{
  // Read the input commands from Serial3
  if(Serial3.available() > 0)
  {
    delta_req = Serial3.parseFloat();
    velocity_req = Serial3.parseFloat();
    if (Serial3.read() == '$') //done transmitting
    {
      angle_req = delta_req;
      angle_req = constrain (angle_req,-32, 32);
```

```
      velocity_req = constrain (velocity_req,-0.8, 0.8);
      if (velocity_req >=0)
      {DriveMotorValue = -(velocity_req*72.504+4.254);} // calibration from
          velocity commands to motor control values
      else
      {DriveMotorValue = -(velocity_req*72.701-3.8011);}
      if (DriveMotorValue <=5 && DriveMotorValue >= -5)
      {DriveMotorValue = 0;}

      potRest = (double)analogRead(SteerPot); // read the steering pot
         position
      angle_act = angleCali(potRest); // calibration from steering pot
         readings to steering angles
      processMotor2(angle_req); // steering motor action
      DriveMotor.motor(1, DriveMotorValue); // driving motor action
      PrintServoPosition(angle_req);
     }
   }
    else // if Serial3 == 0, stop the motors
    {
      DriveMotor.motor(1, 0);
      DriveMotor.motor(2, 0);
    }
}


// Steering motor PID control
void processMotor2(double angle_req)
{
  double potRest = (double)analogRead(SteerPot);
  double angle_act = angleCali(potRest);
  double gap = abs(angle_req - angle_act);
  if(abs(gap) < 3 || abs(gap) > 50 ) // if steering error < 3 or > 50, the
     steering motor does not response
  {
    myPIDSteer.SetTunings(0, 0, 0);
  }
  else
  {
    myPIDSteer.SetTunings(consKp, consKi, consKd);
  }
  myPIDSteer.Compute(); // compute steering motor control values

  digitalWrite(SAB_ESTOP, HIGH);
  DriveMotor.motor(2, SteerMotorValue);
  // Serial.println(SteerMotorValue);
}

// Calibration of the SteerPot readings to front wheel steering angles
double angleCali(double x) // x is the reading of the SteerPot A0 (463~705)
{
  double y = 0.26446 * x - 154.446;
  return (double)y; // return the current angle y, in degrees
}

// Print requested steering commands, actual steering angles, and errors
// onto a Hyper Terminal program on the host computer. This is for data
```

```
// saving and analysing purposes.
void PrintServoPosition(double RequestedAngle)
{
  double potRest = (double)analogRead(SteerPot);
  double currentAngle = angleCali(potRest);
  double angleError = RequestedAngle - currentAngle;

  //Serial.print("Requested Angle: ");
  Serial.print(RequestedAngle);
  Serial.print(" ");
  //Serial.print("  Real Position:");
  Serial.print(currentAngle);
  Serial.print(" ");
  //Serial.print("  Error:");
  Serial.println(angleError);
  //delay(400);
}

void allStop() // stop all motors
{
  int motorZero = 0;
  //Serial.println("All Stop");
  digitalWrite(SAB_ESTOP, LOW);
  DriveMotor.motor(1,motorZero);
  DriveMotor.motor(2,motorZero);
}
```