ARES
The Autonomous Robotic Environmental Sensor

by
Benjamin M. Dyer

A Thesis
presented to
The University of Guelph

In partial fulfillment of requirements
for the degree of
Masters of Applied Science
in
Engineering

Guelph, Ontario, Canada

ABSTRACT

ARES

THE AUTONOMOUS ROBOTIC ENVIRONMENTAL SENSOR

Benjamin M. Dyer

University of Guelph, 2021

Advisors:

Dr. Mohammad Biglarbegain

Dr. Amir Aliabadi

Sensing the indoor environment is a complicated task accomplished through the use of multiple expensive sensors. In order to reduce costs, the Autonomous Robotic Environmental Sensor (ARES) is developed. ARES is a custom-designed omniwheel robot with high modularity, allowing many environmental sensors to be mounted to it. For navigation of an indoor environment, feedback linearization and sliding mode controllers are developed using a kinematic model, modified to include wheel slip. Testing shows the sliding mode controller provides the best performance for indoor environmental sensing.

ARES is used to measure environmental variables in a large laboratory. Measurements are collected at multiple positions periodically over a diurnal cycle. Environmental statistics and predictions of thermal comfort derived from collected data are presented, showing that ARES is capable of taking meaningful measurements of the environment at a fraction of the cost of stationary sensors.

ACKNOWLEDGEMENTS

My thanks to both Dr. Biglarbegian and Dr. Aliabadi for their continuous teaching, guidance, and support throughout my masters. I thank Dana and my family for their continuous support throughout my education. I thank Trevor Smith for being around to bounce ideas off and encouraging me to always work hard. I thank Elyse Hill for always answering controls questions and helping immensely with initial edits. Finally, my thanks goes out to Andrew Newton and the rest of the ICE Lab, AICV Lab, and AIR Lab members, who were always there to teach me topics I was new to.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ARES	Autonomous Robotic Environmental Sensor
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
CAD	Computer Assisted Design
CoR	Center of Rotation
CFD	Computational Fluid Dynamics
C-space	Configuration space
FIFO	First In First Out
GA	Genetic Algorithm
GPIO	General Purpose Input/Output
HVAC	Heating, Ventilation, and Air Conditioning
I^2C	Inter-Integrated Circuit
IR	Infra-Red
LIDAR	Light Detection And Ranging
MRT	Mean Radiant Temperature
PMV	Predicted Mean Vote
PID	Proportional-Integral-Derivative
PPD	Predicted Percent Dissatisfied
PSO	Particle Swarm Optimization
PWM	Pulse Width Modulation
RE	Random Error

RH	Relative Humidity
RMSE	Root Mean Squared Error
SE	Systematic Error
SLAM	Simultaneous Localization and Mapping
SMC	Sliding Mode Control
TKE	Turbulent Kinetic Energy
VGRAPH	Visibility Graph

SYMBOLS

b	Motor viscous friction	$\mathrm{N\,m\,s}$
$\mathbf{e}$	Error vector	
f_{cl}	Clothing area factor	
h_c	Coefficient of convection	$\mathrm{W\,m^{-2}\,K^{-1}}$
i	Current	A
I_{cl}	Clothing level	clo
J	Moment of inertia	$\mathrm{kg\,m^2}$
k	Turbulent kinetic energy	$\mathrm{m^2\,s^{-2}}$
K	Motor constant	$\mathrm{N\,m\,A^{-1}}$
l	Wheel to robot center distance	m
L	Thermal Load	
	Inductance	H
M	metabolic rate	met
p_a	Water vapor pressure	Pa
r	Robot wheel radius	m
R	Resistance	Ω
S	Wind speed	$\mathrm{m\,s^{-1}}$
$\mathcal{T}$	Period	s
T	Temperature	°C or K
t	Time	s
t_a	Ambient air temperature	K
t_{cl}	Temperature of a clothed surface	K
t_L	Large eddy turnover time	s
$\bar{t}_r$	Mean radiant temperature	K
U	Wind velocity component in the x direction	$\mathrm{m\,s^{-1}}$
$\mathbf{U}$	Time-mean Eulerian velocity	$\mathrm{m\,s^{-1}}$
v_s	Slip velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{v}_s$	Slip velocity vector	$\mathrm{m\,s^{-1}}$
v_{wheel}	Wheel translational velocity	$\mathrm{m\,s^{-1}}$
V	Wind velocity component in the y direction	$\mathrm{m\,s^{-1}}$
	Lyapunov function	
	Voltage	V
W	Work	met
	Wind velocity component in the z direction	$\mathrm{m\,s^{-1}}$

GREEK SYMBOLS

α	Angular position of wheel w.r.t. the robot	rad
β	Angular offset	rad
γ	Angular position of rollers	rad
ϕ	Robot wheel angular position	rad
Λ	Eulerian integral length scale	m
σ	Sliding surface (manifold)	
τ_i	Eulerian integral time scale	s
θ	Angular position	rad
$\boldsymbol{\xi}_I$	Robot state vector in the global frame	
$\boldsymbol{\xi}_d$	Robot desired state vector in the global frame	

LIST OF APPENDICES

Chapter 1

Introduction

Environmental sensing encompasses a vast range of measurement types and scales. Measurements may include wind velocities, temperature, soil moisture, long and short wave radiation, humidity, sound levels, pollution, and more. Measurements also range drastically from the small-scale measurements of airspeed and temperature in a duct to large-scale measurements of wind, temperature, and humidity profiles throughout the atmosphere. Remote sensing aside, measurements of the atmosphere, both indoors and outdoors, are accomplished using sensor stations. By using multiple stations, a larger area of the environment can be measured. However, to make accurate spatial measurements, a dense sensor network is required, which is often impractical due to both the monetary cost to set up and environmental changes due to the sensors. As a replacement to dense sensor networks, a sensor station can be mounted on a mobile platform in order to gain high spatial resolution at the expense of temporal resolution.

In order to build a mobile sensing station, an appropriate mobile platform needs to be developed. Some sensors require knowledge of their orientation (in particular, anemometers that detect wind direction); therefore, a platform that can maintain angular position while traversing the environment is useful. Most land-based robots cannot accomplish this as they must turn to change directions; however, omniwheel robots do not suffer from this constraint, making them an

improved option for a mobile sensing station.

The following section will review previous work on omniwheel robots, methods to control them, path planning algorithms, mobile environmental sensors, mapping of indoor environments, and methods for determining thermal comfort.

1.1 Literature Review

1.1.1 Omniwheel Robots

Omniwheel robots utilize omnidirectional wheels, making them holonomic, allowing the robot to utilize each degree of freedom separately (Borisov et al., 2015). Several wheel designs have been proposed and tested, with Swedish and Mecanum omniwheels being the most readily available. Both wheel types use a main wheel with several smaller rollers attached. The Mecanum wheel attaches the rollers at 45 degrees to the wheel's plane, while Swedish wheels attach the rollers at 90 degrees to the wheel's plane.

Mecanum wheels, displayed in Figure 1.1, are typically used in a four-wheel configuration (Borisov et al., 2015; Kilin et al., 2017). The Mecanum wheel rollers create a secondary force away from the direction of rolling. Making the in-line wheels' rollers face opposite directions allows for a holonomic configuration. This configuration enables smoother rolling compared to other omniwheels; however, since the robot uses four wheels, suspension is required to guarantee each wheel can provide the same force on the robot and minimize slip. Additionally, the robots' rectangular shape makes them ideal for use in swarms, where the robots can form convoys and blocks to work together to move large and heavy objects (Lin and Shih, 2013). A major disadvantage of the Mecanum wheel is the inefficiency it introduces to the robot. Since the wheels' force vectors are always facing different directions, the wheels are always working against each other, increasing power consumption and decreasing efficiency.

Figure 1.1: Vex Robotics Mecanum wheel[1].

The 90-degree Swedish wheel, seen in Figure 1.2, can be configured to provide better energy efficiency when compared to Mecanum wheels (Liu et al., 2007). Placing four Swedish wheels radially about the robot makes it possible to create a holonomic configuration capable of utilizing the entire force produced by two wheels by allowing the other two wheels to act as passive rollers. In fact, if two opposing wheels are made completely passive, the robot becomes a type of differential drive robot (Zobova and Tatarinov, 2008). The use of four wheels requires the implementation of suspension in order to guarantee all wheels will provide an equal force on the robot. Suspension can be avoided by using three Swedish wheels equally spaced radially about the robot; however, this comes at the cost of efficiency since any translational movement will require some of each of the wheels' force vectors to counteract each other. A major issue with the Swedish wheel is an increase in vibrations and slippage due to the roller design creating avoidable non-linearities and injecting noise in the system, which must be accounted for when describing the system dynamics (Stonier et al., 2007; Bramanta et al., 2017).

[1] https://www.vexrobotics.com/mecanum-wheels.html

Figure 1.2: Vex Robotics 4" Swedish wheel[2].

1.1.2 Controllers

All forms of omniwheel robots are non-linear systems, requiring the implementation of non-linear control strategies. A large number of controllers have been developed for different configurations of omniwheel robots. As will be discussed in greater detail in Chapter 4, the main controller considered in this thesis is a sliding mode controller. Therefore, several studies with this control design were reviewed.

Sun et al. (2020) developed a non-singular terminal sliding mode controller for a four-wheel-drive Mecanum wheel omnidirectional mobile robot to achieve path-tracking. The controller was able to achieve superior tracking precision and higher robustness compared to conventional sliding mode control. However, it lacked the ability to account for drift due to slip and had increased chatter compared to conventional sliding mode control.

[2]https://www.vexrobotics.com/omni-wheels.html

Alakshendra and Chiddarwar (2016) developed an adaptive sliding mode controller for a four-wheel-drive Mecanum wheel omnidirectional mobile robot using both kinematic and dynamic models of the robot. The adaptive control is able to provide acceptable tracking when sensor fusion between visual tracking and an on-board inertial measurement unit is used. The control scheme is also able to reduce chatter in comparison to a conventional sliding mode controller.

Zhang et al. (2019) focused on developing energy-optimal motion control for a four-wheel Mecanum mobile robot. A power consumption model is proposed in order to determine and reduce power consumption. Through simulation and experimentation, the model was shown to be over 95% accurate in simple scenarios; however, more investigation is required for robots moving on an incline and with an offset center of gravity due to load.

A major issue with the control of omniwheel robots is the assumption that the force of the wheel originates from the center of the wheel; however, this is usually not the case as the contact point deviates from the center of the wheel throughout each rotation. De Villiers and Tlale (2012) developed a control model for a four-wheel Mecanum omniwheel mobile robot which took modelling uncertainties into account. Using this model, system performance was enhanced both for a single Mecanum wheel and a four-wheel Mecanum platform.

Stonier et al. (2007) developed non-linear slip dynamics to describe a three-wheel Swedish wheel mobile robot. A Proportional-Derivative (PD) controller and a direct torque controller were developed, which in conjunction with the non-linear slip dynamics, allowed an increase in performance on a variety of surfaces with different frictional coefficients.

1.1.3 Mobile Indoor Environmental Sensing

Measurement of the indoor environment has proven to be extremely important when creating energy-efficient buildings and comfortable environments for the people within them. As noted by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE),

temperature, humidity, air quality, light levels, and acoustics are all important conditions that must be controlled indoors (Haberl et al., 2008). The environmental variables vary widely within a building, both spatially and temporally, making it important to constantly monitor the environment for accurate control (Zhang et al., 2013; Schiavon et al., 2017; Bulińska et al., 2014). Environmental sensing is typically accomplished using a set of stationary sensors positioned strategically within a building. A lack of accurate, easy to use, and inexpensive sensors reduces scalability when using stationary sensors (Heinzerling et al., 2013). In addition, the use of stationary sensors allows for high temporal resolution but low spatial resolution making it more difficult to predict the environmental variables throughout a building without high monetary and energy costs (Jin et al., 2018).

It is possible to alleviate the issue of low spatial resolution, at a relatively low cost, through the use of mobile sensing (Heinzerling et al., 2013). Since the 1980s, sets of sensors have been mounted on portable carts and moved about buildings to map the environment (Schiller et al., 1988). These mobile sensor carts would be moved to set locations in the building where they would take measurements of the environment (Nicole and McCartney, 2000; Benton et al., 1990). While the sensor carts adequately accomplished reducing sensor costs, they are impractical for continuous extended use as a human is needed to move them around. One solution is to install the sensors on top of a mobile robot allowing for the removal of human interaction.

Several mobile sensing solutions have been proposed. For sensing of air contamination, Chen et al. (2017) developed a method using multiple sensing robots to determine the location of a time-varying contamination source. Fukazawa and Ishida (2009) outfitted an RC car with gas sensors in order to locate a gas source. Widyantara et al. (2018) designed a differential drive robot featuring an anemometer alongside gas sensors to determine the location of gas leaks quickly. An outdoor pollution monitoring system developed by Reggente et al. (2010) was able to measure NO_2, O_3, CO_2, PM10, temperature, and humidity. More recently, a sensing robot created by Jin et al. (2018) utilized a differential drive robot to monitor and create spatial maps of the indoor air quality.

1.1.4 Path Planning

It is imperative to take measurements of environmental variables within short periods of time (e.g. half an hour) to ensure the environment does not change significantly during the measurement period. Naturally, the more efficient the robot's path planning, the larger the area that can be sensed over a given period. In addition, efficient path planning can lower the robot's power consumption, resulting in longer battery life and a lower impact on the environment being measured.

Path planning methods are split into two major categories: offline, where the environment is known, and path planning can occur prior to execution, and online, where the environment is partially or completely unknown and path planning must happen in real-time using the information gathered by the robot as it travels (Raja and Pugazhenthi, 2012).

Within the category of offline path planning, there are two main approaches: classic and evolutionary. Many of the classic approaches are based on the configuration space (C-space) proposed by Udupa and Murthy (1977) and developed by Lozano-Pérez and Wesley (1979). This method treats the robot as a point and increases the size of all obstacles to compensate for the robot's size, reducing path planning to a 2-dimensional problem.

Lozano-Pérez and Wesley (1979) used the C-space in conjunction with a visibility graph (VGRAPH) created by drawing lines between polygon objects within line of sight of each other and where the drawn lines were between the start and end points of the robot. The graph was then used to determine the shortest path between the start and end points. This method is effective in sparse environments, but it begins to suffer from long computational time in dense environments as the number of lines between objects increases (Siegwart et al., 2011). Other mapping methods have been proposed; however, they generally do not provide shorter paths than the VGRAPH method (Raja and Pugazhenthi, 2012).

The C-space has also been used by Lozano-Pérez (1983), Zhu and Latombe (1991), Likhachev and Ferguson (2009), and many others by breaking the space down into a grid in which each cell

either contains an obstacle or does not. Using this grid, paths through free cells can be found in order to determine the shortest path. While this method is fast for low resolutions, approaching the optimal path requires a finer mesh, which exponentially increases the computational cost.

The classical methods for path planning, while effective, are slow at determining collision-free paths and tend to only provide locally optimal solutions without finding the globally optimal solution. In addition, Canny and Reif (1987) have shown that the problem is non-deterministic polynomial-time hard. To handle the possibility of a more complex environment and even an environment containing dynamic obstacles, a multitude of evolutionary algorithms have been developed.

Genetic Algorithm (GA), based on natural selection, selects a set of possible paths and then evolves them by making minor changes and finding the new most efficient paths (Holland, 1992). The GA method can be combined with other path planning techniques such as classical approaches where near-optimal paths are found using a different approach and then improved using GA (Dozier et al., 1997). AL-Taharwa (2008) used fixed-length paths to find a solution; however, in complex environments, the method can take hours to find a solution.

Particle Swarm Optimization (PSO) is a widely-used method based on birds' and fish's social behaviour. PSO is simpler in implementation due to fewer parameters (Kennedy and Eberhart, 1995). Similar to the GA, PSO has been combined with a graph-based approach in Qin et al. (2004), first to find collision-free paths and then use the resulting paths as the starting path for the PSO to optimize.

Evolutionary approaches suffer from non-smooth path planning, increasing the tracking error. In this thesis, a classical approach will be used, specifically the D*-lite algorithm. D*-lite searches for the optimal path in C-space and can handle both static and dynamic environments and can be implemented in offline and online modes (Koenig and Likhachev, 2002). This flexibility makes it ideal for the platform that will be developed, as it is simple to modify for use in highly populated and dynamic indoor environments.

1.1.5 Environmental Mapping

There is a lack of research regarding the mapping of indoor wind speed and relative humidity, which greatly affect thermal comfort (Liu et al., 2016).

Measurements of turbulent variances and fluxes are conducted using static sensors at strategic positions, reducing systematic and random errors by measuring over a long period of time. However, since this requires multiple sensors to measure a large space, a trade-off can be made using a single mobile sensor that measures over a shorter period at each location. As discussed by Lenschow et al. (1994) and Aliabadi et al. (2016) the systematic error and random error can be described, respectively, as

$$SE = 2\frac{\tau_i}{\mathcal{T}} \tag{1.1}$$

$$RE = \sqrt{2\frac{\tau_i}{\mathcal{T}}} \tag{1.2}$$

where τ_i is the Eulerian integral time scale and $\mathcal{T}$ is the measurement period. Here it is assumed that $\tau_i \ll \mathcal{T}$. It is clear from these equations that the errors are reduced as the measurement period increases, and the measurement period must be significantly longer than τ_i in order to gain meaningful measurements. As such, it is important to know the typical time scale for the indoor environment.

Experiments conducted by Puits et al. (2013) show the time scale for temperatures are on the order of a few minutes while the eddy turnover time is on the scale of tens of seconds in a small room. Spilak et al. (2016) showed that the time scale for turbulence indoors could be on the order of a few seconds. Takimoto et al. (2011) gave a more general eddy turnover time based on environment size corresponding to a few seconds to tens of seconds. Since previous research uses eddy turnover time and not integral time scale, a relationship between them must be determined.

1.1.6 Thermal Comfort

Thermal comfort is the measure of an individual's perceived physical comfort based on the environment around them. Due to reliance on an individual's thermal sensations, thermal comfort is intrinsically subjective. The ASHRAE thermal sensation scale has been developed in order to quantify thermal comfort. The scale ranges from -3 to 3, with 0 being a neutral sensation, negative values being cold, and positive values being hot (ASHRAE, 2017). Normally thermal comfort would be quantified on the scale by surveying occupants of a building; however, it is useful to determine an average thermal sensation rating for an environment based on quantitative measurements of environmental physical variables.

Fanger (1972) developed the widely used Predicted Mean Vote (PMV) and the Predicted Percent Dissatisfied (PPD) models. The PMV-PPD model uses four environmental variables (ambient air temperature, relative humidity, wind speed, and mean radiant temperature) and two physiological variables (metabolic rate and clothing insulation) to predict the average thermal comfort vote of a large group of people in a given environment. While environmental variables can be measured or estimated, physiological variables are determined using the ASHRAE thermal comfort standard, which provides tables of metabolic rates and clothing levels during different activities and for different types of clothing. Mean Radiant Temperature (MRT) can be estimated based on the ambient air temperature since the MRT and ambient temperature have a mean difference of 0.3 K and a median absolute difference of 0.4 K. These differences are based on an analysis of the ASHRAE Global Thermal Comfort Database, five field studies, and five laboratory test conditions (Dawe et al., 2020; Földváry Ličina et al., 2018). Environmental variables can be numerically simulated using Computational Fluid Dynamics (CFD) or other models or measured in the environment.

CFD simulations have been used extensively to predict thermal fields, airflow patterns, concentration distributions of gasses and particulates, and prediction of thermal comfort (Nielsen, 2015; Zhao et al., 2003). However, a major drawback of CFD is the reliance on initial and boundary con-

ditions, particularly wall surface temperatures, air inlet temperatures, and inlet and outlet airflow rates. Small differences in these conditions can significantly change simulation results, with errors upwards of 20% (Posner et al., 2003). In addition, CFD calculations are complex, require computing facilities, and are often beyond the skills of the most practical engineers (Aliabadi et al., 2011). A combination of experimental data collection and numerical simulations can produce more accurate results by using collected data as boundary conditions or for verification purposes of the simulation (Shan and Lu, 2020). In an ideal world, a dense sensor network can be assembled, resulting in highly-accurate spatial and temporal measurements of the environment; however, dense sensor networks can interfere with occupants using the space, changing the environment. In addition, dense sensor networks are prohibitively expensive both in set-up and maintenance, depending on sensor type (Williams, 2019). Therefore, it is desirable to collect environmental data using a non-intrusive, low-cost method. One potential solution is the use of mobile robots to autonomously transport sensor stations about the environment resulting in higher spatial resolution data sets at the cost of lower temporal resolution (Jin et al., 2018).

1.2 Research Gaps

A review of the literature reveals a number of gaps. Most controllers developed for omniwheel robots are either designed for four-wheels or are proposed but not implemented on physical systems. Of these controllers, only a few use dynamic models including wheel slip, and to the best of the author's knowledge, no known controllers have been developed using a kinematic model with slip.

Most mobile environmental sensing platforms have been developed for gas source localization purposes, while sensing of the environment for thermal comfort has been limited to sensor carts or static sensor stations. Few mobile robots have been developed to measure wind velocity and relative humidity, and to the best of the author's knowledge, no platforms have been designed to

predict thermal comfort. In addition, omniwheel robots have not been used for environmental sensing, despite the possible advantages provided by a holonomic platform, opening up the possibility of developing an omniwheel based environmental sensing robot with a focus on measuring thermal comfort in the indoor environment.

1.3 Objectives

This thesis aims to develop and prototype an omniwheel robot platform that will be able to carry out environmental sensing experiments. In order to accomplish this task, the following objectives must be met:

- Develop an omniwheel robot platform capable of transporting an array of environmental sensors and supporting hardware.
- Develop a kinematic model of the robot platform, which includes wheel slip.
- Develop and test non-linear position controllers to determine the most effective controller for the robot.
- Design and carry out an indoor environmental sensing experiment over a full diurnal cycle in an office environment, focusing on the prediction of thermal comfort.

1.4 Structure of the Thesis

The thesis is structured as follows. Chapter 2 provides the necessary mathematical background for this thesis. Chapter 3 discusses the electrical and mechanical development of the robot as well as characterization of the robot's intrinsic properties. Chapter 4 develops models for the robot and its wheels, develops controllers for the robot, and includes experimental data, which is analyzed to determine the most useful controller for environmental sensing. Chapter 5 discusses the path

planning and environmental experiments that were carried out and analyzes the environmental data. Chapter 6 concludes the thesis and provides a path for future work on the platform.

Chapter 2

Background

Before developing the Autonomous Robotic Environmental Sensor (ARES) and deploying it for experiments, background knowledge must be reviewed. This chapter will discuss the mathematics behind useful linear and non-linear controllers before defining several environmental statistics. Finally, the Predicted Mean Vote and Predicted Percent Dissatisfied (PMV-PPD) models are defined.

2.1 Controllers

When developing controllers, it is important first to define the system using a state-space representation. The state-space equations contain the dynamics or kinematics of the system, including the controller. For linear systems, the state-space equations are described generally as

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{2.1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \tag{2.2}$$

where $\mathbf{x}(t)$ is the state vector, $\mathbf{y}(t)$ is the output vector, $\mathbf{A}$ is the system matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the output matrix, $\mathbf{D}$ is the feed forward matrix, and $\mathbf{u}(t)$ is the input vector which is set by the controller. The linear state-space equations are necessary for accurately controlling a linear system, such as a DC motor. However, if the system is non-linear a more generalized state-space representation is defined as

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t)\right) \tag{2.3}$$

$$\mathbf{y}(t) = \mathbf{h}\left(\mathbf{x}(t), \mathbf{u}(t)\right), \tag{2.4}$$

where $\mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t)\right)$ and $\mathbf{h}\left(\mathbf{x}(t), \mathbf{u}(t)\right)$ are nonlinear functions based on the kinematics or dynamics of the system. If both functions can be described as linear combinations of state and input variables the system can instead be described using (2.1) and (2.2).

2.1.1 PID Controller

A proportional-integral-derivative (PID) controller is an extremely popular method for controlling linear systems. The controller uses a feedback scheme that subtracts the desired state of a system from its measured state resulting in the system error, $\mathbf{e}(t)$. The error is then fed into the control function defined as

$$\mathbf{u}(t) = \mathbf{K}_p\mathbf{e}(t) + \mathbf{K}_i \int_0^t \mathbf{e}(t')dt' + \mathbf{K}_d \frac{d}{dt}\mathbf{e}(t) \tag{2.5}$$

where $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x_d}(t)$ is the system error, $\mathbf{x_d}(t)$ is the desired trajectory of the system, and $\mathbf{K}_p$, $\mathbf{K}_i$, and $\mathbf{K}_d$ are the proportional, integral, and derivative design matrices, respectively. It is often unnecessary to include all three terms to obtain desired system behaviour, resulting in PI, PD, or P controllers depending on which terms are included.

When tuning a PID controller, it is important to understand how each gain affects the system response. The proportional gain, $\mathbf{K}_p$, multiplies the error, which in turn reduces error over time. The larger the proportional gain, the quicker the system will reach its desired state. However, a large proportional gain can cause the system to overshoot the desired state resulting in oscillations. If the proportional gain is too large, the oscillations will destabilize the system causing unpredictable behaviour.

The derivative gain, $\mathbf{K}_d$, can help reduce oscillations about the desired state as it acts to slow the rate of change of the system's state. The derivative term effectively acts as a damping force that can be tuned to cause the controller to be over-damped, critically-damped, or under-damped as desired. Since the derivative term changes based on the error's rate of change, it cannot be used as a controller by itself and always requires either a P or I term to be included in the controller.

The integral gain, $\mathbf{K}_i$, affects the steady-state error. If the system contains a small, constant error, the integral of the error increases in magnitude, increasing the integral term and causing the error to shift towards the desired state. A large integral term will cause oscillations about the desired state since the integration causes the term to be time delayed. In most cases, it is not useful to integrate over all time, so the error is normally integrated from the near past to the current time (often on the scale of a few seconds).

2.1.2 Feedback Linearization

When handling non-linear systems, a PID controller no longer provides accurate and reliable control. One of the simplest methods for handling non-linear systems is feedback linearization. Feedback linearization uses a transformation to convert a non-linear system into a linear system, at which point a PID controller can be applied. In order to apply feedback linearization, the system

must be representable in control-affine form, as

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(t, \mathbf{x}(t)\right) + \mathbf{g}\left(t, \mathbf{x}(t)\right)\mathbf{u}(t), \tag{2.6}$$

where $\mathbf{f}\left(t, \mathbf{x}(t)\right)$ and $\mathbf{g}\left(t, \mathbf{x}(t)\right)$ are linear or non-linear functions of the state vector. The controller $\mathbf{u}$ is then defined as a function of the linear controller $\mathbf{v}(t)$ as

$$\mathbf{u}(t) = \mathbf{a}\left(t, \mathbf{x}(t)\right) + \mathbf{b}\left(t, \mathbf{x}(t)\right)\mathbf{v}(t). \tag{2.7}$$

where $\mathbf{a}\left(t, \mathbf{x}(t)\right)$ and $\mathbf{b}\left(t, \mathbf{x}(t)\right)$ are design functions.

The goal is for the controller $\mathbf{u}(t)$ to render a linear mapping between the new controller $\mathbf{v}(t)$ and the state vector $\dot{\mathbf{x}}(t)$. This mapping can be obtained fairly easily by defining $\mathbf{a}\left(t, \mathbf{x}(t)\right) = -\mathbf{f}\left(t, \mathbf{x}(t)\right)$ and $\mathbf{b}\left(t, \mathbf{x}(t)\right) = \mathbf{g}^{-1}\left(t, \mathbf{x}(t)\right)$ resulting in the transformed state-space equation

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t). \tag{2.8}$$

Any linear control method can be used to design $\mathbf{v}(t)$. It should be noted that while feedback linearization is a useful tool when dealing with non-linear systems, it cannot handle non-linear systems with uncertainty. As such, it is useful to have other methods for developing non-linear controllers, such as sliding mode control.

2.1.3 Sliding Mode Control

Sliding mode control applies a discontinuous signal to a system causing the system trajectories to "slide" along a "sliding surface". The controller consists of two parts, a continuous control signal to push the system towards the sliding surface and a discontinuous signal to keep the system on the sliding surface. SMC can be applied to control-affine systems of the form (2.6) where $\mathbf{f}(t, \mathbf{x}(t))$ and $\mathbf{g}(t, \mathbf{x}(t))$ are continuous and smooth.

To create a sliding mode controller, a sliding surface (a manifold) is chosen such that the system acts as desired when on the surface. Feedback gains must be chosen such that the system can reach and stay on the sliding surface. The sliding surface is defined by a switching function $\sigma(\mathbf{x}(t))$ which is analogous to the distance to the sliding surface from the position $\mathbf{x}(t)$. By definition of the sliding surface, when $\mathbf{x}(t)$ is not on the sliding surface $\sigma(\mathbf{x}(t)) \neq 0$ and when $\mathbf{x}(t)$ is on the sliding surface $\sigma(\mathbf{x}(t)) = 0$. The sliding surface is n dimensioned where n is the number of controllable states and can be described as

$$\{\mathbf{x}(t) \in \mathbb{R}^n : \sigma(\mathbf{x}(t)) = 0\}. \tag{2.9}$$

For the system to reach the sliding surface, the system must reach $\sigma(\mathbf{x}(t)) = 0$ from any initial condition and once at $\sigma(\mathbf{x}(t)) = 0$ the controller must be able to keep the system on the sliding surface at all times.

To show the sliding mode exists, consider a simple Lyapunov function

$$V(\sigma(\mathbf{x}(t))) = \frac{1}{2}\sigma^T(\mathbf{x}(t))\sigma(\mathbf{x}(t)), \tag{2.10}$$

where $\sigma^T(\mathbf{x}(t))\sigma(\mathbf{x}(t)) = ||\sigma(\mathbf{x}(t))||_2^2$ is the distance to the sliding surface. Asymptotic stability of the sliding surface exists and is stably reachable when the Lyapunov function is negative resulting in

$$\sigma^T(\mathbf{x}(t))\sigma(\mathbf{x}(t)) < 0. \tag{2.11}$$

Given the simple relationship in (2.11), stability can be guaranteed by constraining $\sigma(\mathbf{x}(t))$ as

$$\begin{cases} \dot{\sigma}(\mathbf{x}(t)) < 0 & \text{when } \sigma(\mathbf{x}(t)) > 0 \\ \dot{\sigma}(\mathbf{x}(t)) > 0 & \text{when } \sigma(\mathbf{x}(t)) < 0. \end{cases} \tag{2.12}$$

Note that this condition only guarantees system stability when $\sigma(\mathbf{x}(t)) \neq 0$ and $\dot{\sigma}(\mathbf{x}(t)) \neq 0$. Around $\sigma(\mathbf{x}(t)) = 0$, the controller is discontinuous allowing it to switch over the 0 condition keeping the controller stable at all times.

2.2 Environmental Variables

Analysis of the environmental variables requires statistics to be executed. These statistics are averages, variances, and covariances (fluxes). Here we define the statistics and how they are calculated for later use.

All environmental variables can be split into two parts, the average value over a measurement period and the random fluctuation of the value, known as Reynolds decomposition. Mathematically, for a variable X, this is expressed as

$$X = \overline{X} + x, \tag{2.13}$$

where $\overline{X}$ is the average value of X and x is the random variation in X at the time of measurement. For a set of N discreet measurements, the average of a variable is calculated as

$$\overline{X} = \frac{1}{N} \sum_{i=1}^{N} X_i. \tag{2.14}$$

With the average defined, the variance of a variable, $\overline{x^2}$, can also be defined as

$$\overline{x^2} = \frac{1}{N}\sum_{i=1}^{N}\left(X_i - \overline{X}\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_i\right)^2. \tag{2.15}$$

It can be seen from (2.15) that the variance of X is the squared average of its random fluctuations. The covariance between two variables can be defined similarly. The covariance of X and Y is defined as

$$\overline{xy} = \frac{1}{N}\sum_{i=1}^{N}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right) = \frac{1}{N}\sum_{i=1}^{N} x_i y_i \tag{2.16}$$

where $\overline{xy}$ is the covariance between X and Y.

2.2.1 Environment Specific Variables

There are a few important variables to measure and calculate in the environment. Measurement of U, V, and W correspond to the wind velocity in the x, y, and z directions, respectively. In addition, the temperature T and relative humidity RH can be measured. The variances of U, V, W, and T are useful; however, it is beneficial to normalize them. To do so, we define the total average wind speed in the environment and the maximum difference in temperature, respectively, as

$$\overline{S} = \sqrt{\overline{U}^2 + \overline{V}^2 + \overline{W}^2}, \tag{2.17}$$

$$\Delta T = Max\left(T_1, \cdots, T_N\right) - Min\left(T_1, \cdots, T_N\right). \tag{2.18}$$

The variances and covariances are normalized using (2.17) and (2.18) depending on which variables are being observed. Variances of U, V, W, S and covariances between them are normalized with $\overline{S}^2$, variance in temperature is normalized with $(\Delta T)^2$, and covariance between wind velocity components and temperature are normalized with $\overline{S}\Delta T$.

The last, and arguably most important, statistical term when evaluating the turbulence of an environment is the turbulent kinetic energy (TKE), k. The TKE is calculated as half the sum of the variances of the wind velocity components, defined as

$$k = \frac{1}{2}\left(\overline{u^2} + \overline{v^2} + \overline{w^2}\right). \tag{2.19}$$

As with other variance-based statistics, the TKE is normalized using the squared average of total wind speed $\overline{S}^2$.

2.2.2 Integral Time Scale

In Chapter 1 the eddy turnover time was introduced to determine the systematic and random errors; however, these errors are defined using the integral time scale, so the relationship between the two timescales must be shown. As discussed by Barrett and Hollingsworth (2001), several different integral length scales can be used when characterizing turbulence and shear flow, each with its own advantages and disadvantages. One of the most commonly used integral length scales is derived from the Eulerian integral scale and given by

$$\Lambda = \mathbf{U}\tau_i, \tag{2.20}$$

where Λ is the integral length scale, $\mathbf{U}$ is the time-mean Eulerian velocity, and τ_i is the integral time scale. From here, we can use the definition of the eddy turnover time to find that it is equal to the integral time scale using (2.20), i.e.

$$t_L = \frac{\Lambda}{\mathbf{U}} = \tau_i. \tag{2.21}$$

This shows that the eddy turnover time and integral time scales are equivalent, and therefore, can be used interchangeably when calculating the systematic and random errors.

A standard method for finding the integral time scale is by integrating from $t = 0$ to the first zero-crossing of the auto-correlation or cross-auto-correlation function (Ahmadi-Baloutaki and Aliabadi, 2021). The cross-auto-correlation is defined with the knowledge of the mean and variance of variables X and Y as

$$\hat{R}_{XY}(k) = \frac{\sum_{t=0}^{N-k-1} \left[\left(X_t - \overline{X}_t\right)\left(Y_{t+k} - \overline{Y}_{t+k}\right)\right]}{\left[\sum_{t=0}^{N-k-1} \left[\left(X_t - \overline{X}_t\right)\left(Y_t - \overline{Y}_t\right)\right]\right]^{1/2} \left[\sum_{t=0}^{N-k-1} \left[\left(X_{t+k} - \overline{X}_{t+k}\right)\left(Y_{t+k} - \overline{Y}_{t+k}\right)\right]\right]^{1/2}} \tag{2.22}$$

where $\hat{R}_{XY}(k)$ is the cross-auto-correlation function dependent on the time lag between measurements, k is the time step related to the lag time according to $k = \frac{\text{lag time}}{\Delta T} < N$, N is the total number of measurements made, ΔT is the period between time steps, and the averages at t and $t+k$ are calculated as $\overline{X}_t = \frac{1}{N-k}\sum_{t=0}^{N-k-1} X_t$ and $\overline{X}_{t+k} = \frac{1}{N-k}\sum_{t=0}^{N-k-1} X_{t+k}$. The auto-correlation function can be found simply by using the same variable for X and Y.

2.3 PMV-PPD Model

As discussed in Section 1.1.6, the ASHRAE thermal sensation scale can be used to quantify thermal comfort in an environment. Fanger (1972) developed an equation that uses the metabolic rate of a person, the mean skin temperature, and the sweat rate as the physiological parameters which affect the subject's heat balance to predict the mean vote for a large group of people in the environment. A combination of physiological and environmental variables are used to describe a neutral thermal comfort as

$$M - W \longleftarrow 3.96 \times 10^{-8} f_{cl} \left[(t_{cl} + 273)^4 - \left(\overline{t_r} + 273 \right)^4 \right] + f_{cl} h_c \left(t_{cl} - t_a \right)$$
$$+ 3.05 \left[5.73 - 0.007(M - W) - p_a \right] + 0.42 \left[(M - W) - 58.15 \right]$$
$$+ 0.0173M(5.87 - p_a) + 0.0014M(34 - t_a), \tag{2.23}$$

where

$$t_{cl} = 35.7 - 0.0275(M - W) - R_{cl} \left(\, (M - W) - 3.05 \left[5.73 - 0.007(M - W) - p_a \right] \right.$$
$$\left. - 0.42 \left[(M - W) - 58.15 \right] - 0.0173M(5.87 - p_a) - 0.0014M(34 - t_a) \, \right), \tag{2.24}$$

is the temperature of the clothed surface, f_{cl} is the clothing area factor defined as the ratio between the surface area of a clothed body and the DuBois surface area of a nude body, $\overline{t_r}$ is the mean radiant temperature of the room, h_c is the coefficient of convection on the body's surface, t_a is the ambient air temperature, M is the metabolic rate of the person (units of met), W is the external work on the person (i.e. external heating or cooling), and p_a is the water vapour pressure in ambient air.

The coefficient of convection, h_c, and clothing area factor, f_{cl} can be estimated using tables provided in ASHRAE (2017) or found using the following relations:

$$h_c = \begin{cases} 2.38(t_{cl} - t_a)^{0.25} & 2.38(t_{cl} - t_a)^{0.25} > 12.1\sqrt{S} \\ 12.1\sqrt{S} & 2.38(t_{cl} - t_a)^{0.25} < 12.1\sqrt{S}, \end{cases} \tag{2.25}$$

$$f_{cl} = \begin{cases} 1.0 + 0.2I_{cl} & I_{cl} < 0.5 \text{ clo} \\ 1.05 + 0.1I_{cl} & I_{cl} > 0.5 \text{ clo}. \end{cases} \tag{2.26}$$

where S is the total wind speed, and I_{cl} is the level of clothing being worn as defined in the ASHRAE fundamentals handbook using units of clo.

The predicted mean vote (PMV) estimates the average rating that a large group of people will give to thermal comfort on the ASHRAE thermal sensation scale. The PMV index is defined by the subject's metabolic rate and the thermal load on the subject's body. Fanger (1972) determined an equation for the PMV defined as

$$\text{PMV} = [0.303 \exp(-0.036M) + 0.028]L, \tag{2.27}$$

where L is the thermal load on the subject's body. The thermal load is calculated as the difference between the right and left sides of (2.23). In order to complete the calculation, t_{cl} must be calculated iteratively as

$$\begin{aligned} t_{cl} =& 35.7 - 0.028(M - W) - R_{cl}[39.6 \times 10^{-9} f_{cl}[(t_{cl} + 273)^4 - (\overline{t_r} + 273)^4] \\ &+ f_{cl} h_c (t_{cl} - t_a)]. \end{aligned} \tag{2.28}$$

While the PMV is a useful index, it is often more useful to quantify the percentage of people who will be dissatisfied with their thermal comfort. This is represented by the predicted percent dissatisfied (PPD) index, which is a function of the PMV defined as

$$\text{PPD} = 100 - 95 \exp(-(0.03353\text{PMV}^4 + 0.2179\text{PMV}^2)). \tag{2.29}$$

By definition, the lowest value of the PPD is 5%, and a PMV range of ± 0.5 corresponds to a PPD of 10%. The PMV-PPD model is used widely for quantifying thermal comfort and is defined in the ISO standard 7730[1]with a set of programs to help calculate the PMV and PPD.

[1] https://www.iso.org/obp/ui/#iso:std:iso:7730:ed-3:v1:en

The platform being developed does not use a sensor for measuring the mean radiant temperature, but it uses the ambient air temperature as a proxy. This introduces an average systematic error of ± 0.3 K in estimating the mean radiant temperature (Dawe et al., 2020). This assumption introduces a small error which can be estimated by calculating the largest relative error in PMV and PPD using error propagation. A large error of 0.6 K in mean radiant temperature results in a relative PMV error of $\pm 10\%$ and a relative PPD error of $\pm 13\%$.

Chapter 3

Platform Development

This chapter presents the development of the platform that will be used for control and environmental experimentation.

3.1 System Components

The Autonomous Robotic Environmental Sensor (ARES), shown in Figure 3.1, can be divided into three main subsystems: mechanical, electrical, and sensors. The mechanical subsystem consists of the robot's frame, the wheel drive, and the extra levels on which sensors can be mounted. The electrical subsystem consists of power converters to provide regulated rails at 3.3V, 5V, and 12V and a motor driver board connected to a Teensy 4.0 microcontroller. The sensor subsystem is comprised of a Young81000 ultrasonic anemometer and an HMP60 relative humidity and temperature sensor connected to a Campbell Scientific CR6 data-logger. The rest of this chapter is devoted to an in-depth overview of the platform.

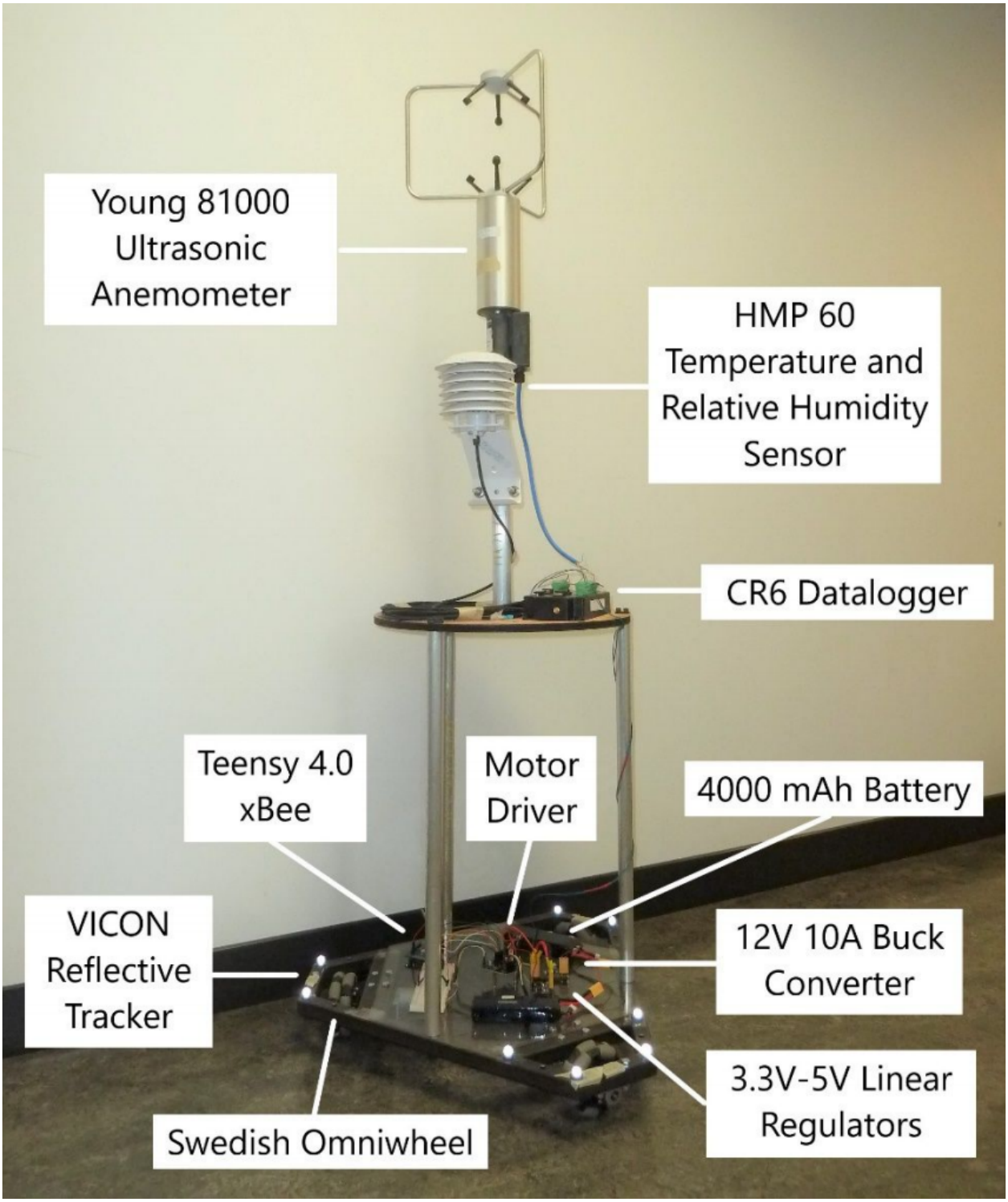

Figure 3.1: ARES set up for thermal comfort measurements.

3.2 Mechanical Subsystem

Omniwheel design influences component selection for the robotic platform. Three of the more prominent design schemes for omniwheels include four-wheel-drive Mecanum wheel robots, four-wheel-drive Swedish wheel robots, and three-wheel-drive Swedish wheel robots.

The four-wheel-drive Mecanum wheel robots provide the smoothest rolling due to each roller having a long contact time with the ground. The longer contact time comes at the cost of the location of contact alternating between both sides of the wheel. Using four wheels results in uneven pressure between the wheels and the ground unless a suspension system is implemented to guarantee all wheels maintain the same contact force. Without all wheels maintaining the same contact force, the robot is prone to slip on slightly uneven terrain, causing drift over time and less accurate control. Since a suspension system would require re-calibration based on robot weight, it is unsuitable for a modular and easily adaptable platform; therefore, a Mecanum wheel design is not considered.

Swedish wheel designs also suffer from variable point of contact; however, the average point of contact will always be at the wheel's center, resulting in less modelling error than produced by Mecanum wheels. The main design choice is between a four-wheel or three-wheel design due to the lower modelling error. In both cases, the wheels are placed radially about the robot's center forming the most stable configuration. The four-wheel design would require a suspension system to be implemented to maintain a reasonable level of controllability, adding complexity to the design. However, the four-wheel design is capable of a higher top speed and is more energy-efficient as moving in straight lines allows it to use only two wheels when properly oriented. The three-wheel design suffers from lower speeds and higher power consumption due to each wheel counteracting the other wheels' force vectors. Depending on the three-wheel design's geometry, the robot's orientation changes the maximum translational speed, which must be accounted for when planning trajectories.

Given the four-wheel design's added complexity, particularly the need for a suspension system, a three-wheel design was chosen. While the three-wheel design is slightly less energy-efficient and has a lower top speed, the extra stability and lower complexity from having only three points of contact with the ground makes it optimal for a modular design. With this design type in mind, the frame of the robot can be developed.

3.2.1 Robot Frame

The robot's most stable configuration is achieved when the wheels are evenly spaced, radially about the center of the robot, with each wheel having the same distance to the robot's center of rotation, as seen in Figure 3.2. This, in turn, means that each wheel is at a 120-degree angle from the other two with respect to the robot's center. The wheel drives are embedded in the robot, allowing the robot to carry higher loads and reducing the risk of damage from the wheels hitting obstacles. This has the added benefit of increasing ARES' maximum load and creating more stable wheels allowing for more accurate control.

The frame was designed around an equilateral triangle, with each side measuring 27.56 inches. The corners of the triangle were cut off such that there would be a 4.33-inch-long segment resulting in a hexagon with alternating 4.33-inch and 18.90-inch side lengths. Finally, three segments were added to the inside of the hexagon such that they were parallel to and 3 inches away from the 4.33-inch segments. The frame's outer segments were built using 1×1-inch square iron tubing, and the inner segments were built using 2×1-inch square iron tubing. All tubing was welded together in order to ensure the frame was rigid and robust. Mounting holes were drilled in the frame as needed.

3.2.2 Wheel Drive

In the interest of allowing any number of sensors to be mounted onto the platform, the wheel drive needed to handle a heavy load. The target load was set to a minimum of $50\,\mathrm{kg}$. The frame

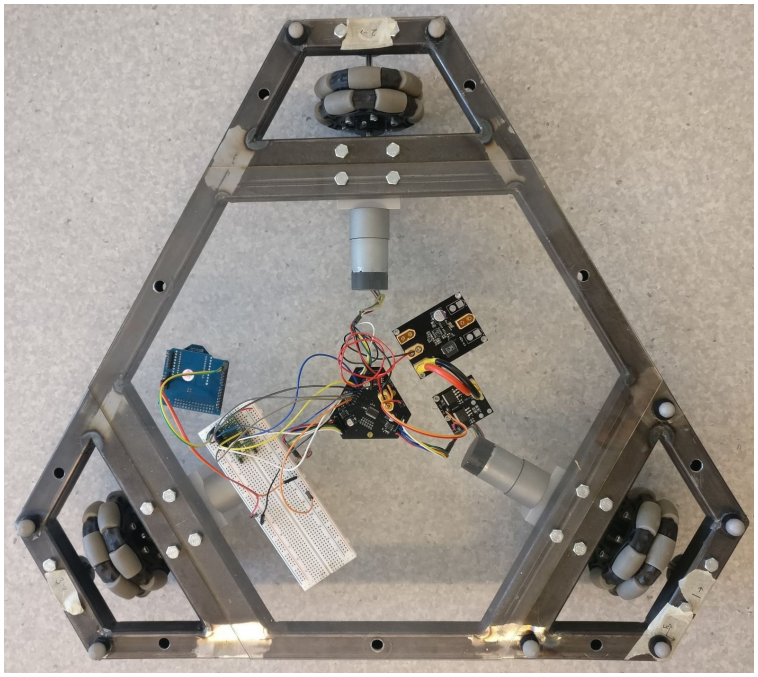

Figure 3.2: Top-down view of ARES' frame with electronics and wheel drives attached.

was designed to support embedded wheels, allowing the axle to be supported on both sides of the wheel. By supporting both sides of the axles, the wheel is guaranteed to always be aligned properly with the ground. Slight misalignment in conventional wheel design poses no significant problem because the wheel has a constant point of contact with the ground. However, misalignment in Swedish wheel robots induces vibrations that affect the wheel drive because the point of contact changes depending on which side of the wheel the roller is on. The vibration issue can be solved by using suspensions; however, since this design does not use suspension, proper alignment is essential.

The wheel drive consists of an axle slotted through two bushings with a wheel mounted between the bushings using wheel hubs on both sides, seen in Figure 3.3. A motor is coupled to the axle using a shaft adapter such that the motor and adapter can be switched, allowing for the motor to be changed quickly, based on the application. The bushings are bolted to the robot frame, and the motor is attached to the frame using an appropriate adapter.

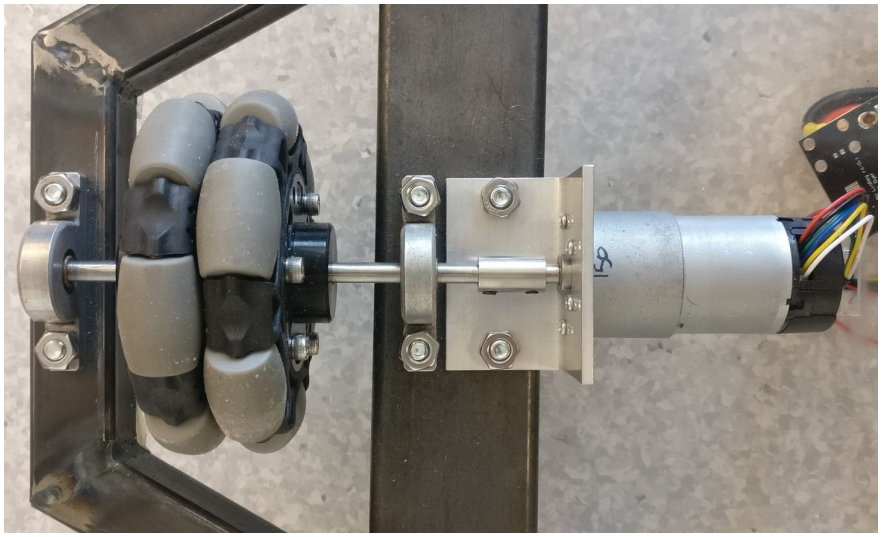

Figure 3.3: View of the wheel drive from the bottom of ARES.

The axle is a 1/4-inch D-shaft allowing for all hardware to be attached to it using set screws. All parts were chosen such that a 1/2-inch axle could be used instead by using alternate parts. Oil-embedded bushings were used instead of ball bearings for ease of assembly and disassembly. Most ball bearings require a push-fit to function properly, but bushings only require a slip fit allowing for easy disassembly at the cost of slightly higher rotational friction. The bushings used are designed to accept a 1/4-inch shaft and can handle shaft misalignment of 5 degrees. The bushings can handle a radial load of 84 kg at 120 rpm, which, when combined with the six bushings, results in a maximum platform weight of 504 kg. An equivalent bushing exists[1] designed to mate with a 1/2-inch axle while maintaining the same mounting hole positions, allowing for an easy swap between 1/4 and 1/2-inch axles without modifying the robot frame.

The omniwheels and wheel hubs were purchased from Vex Robotics. The wheels are 4 inches in diameter and 1.53 inches wide. Each wheel is mounted to the axle using aluminum wheel hubs on each side of the wheel, also purchased from Vex Robotics. Wheel hubs are available to accept either 1/4-inch or 1/2-inch axles. Each hub had a hole drilled and tapped through it so a set screw

[1] https://www.mcmaster.com/5912K4/

could be used to tighten the hub and wheel assembly to the axle.

For environmental sensing, the platform must have high positional accuracy while fast movement is less important. The Pololu 37Dx73L DC motor was chosen for its slow rotational speed of 67 rpm, its high maximum torque of 49 $\mathrm{kg\,cm}$, and the included high-resolution encoder. Given the 5.08-cm radius of the wheels, the maximum linear speed of a single wheel is 0.356 $\mathrm{m\,s^{-1}}$. The encoder measures 64 counts per rotation of the motor shaft resulting in 9600 encoder readings per revolution and a theoretical translational resolution of 33 $\mu\mathrm{m}$. Since the robot will calculate its position based solely on encoders, a high count encoder is extremely important in accurately measuring the robot's location. The motors were mounted to the frame using 1/8-inch angle iron with the necessary mounting holes drilled into them.

3.2.3 Levels

ARES consists of two levels, the bottom level contains all hardware related to the control and powering of the robot, and the upper level is reserved for all equipment related to environmental sensing.

The bottom level was made by laser cutting a piece of 1/8-inch clear acrylic with all necessary mounting holes for electronics. The acrylic was mounted to the frame using the same holes as were used for mounting the motor. Since the motor mounting holes were drilled with loose tolerances, the holes in the acrylic were cut out using a Dremel. This process was repeated for all other holes that the acrylic sheet was covering.

The upper level was mounted 2 feet above the robot frame using three 1-inch diameter aluminum rods positioned radially about the center of the robot. The rods have a single 5/16-inch hole drilled and tapped in the bottom to mount the rods to the robot frame. The tops of the rods have four 11/64-inch holes drilled and tapped. A round piece of plywood was laser cut for the upper level, with appropriate mounting holes such that it could be screwed into the rods. At the center

of the upper level, another set of mounting holes were cut to screw into another rod on which the ultrasonic anemometer, relative humidity, and temperature sensors could be mounted. The top rod was made from a 1.25-inch diameter aluminum rod with four 11/64-inch holes drilled and tapped radially about the rod's face.

3.3 Electrical Subsystem

The electrical subsystem consists of two main components, the power delivery system and the control system. The power delivery system uses a pair of LiPo batteries, a 12V 10A buck converter based on the Analog Devices LTC3807 synchronous step-down controller, and two linear regulators for 3.3V and 5V based on the LM1085 voltage regulator. The control system consists of an X-Bee wifi module, a Teensy 4.0 micro-controller, and a custom motor driver board with feedback from the motor encoders.

3.3.1 Power Delivery

ARES requires 3.3V, 5V, and 12V rails in order to power both the control system and the sensors. Since the motors and most environmental sensors run on 12V, the 12V converter needs to be capable of handling a high power load. As will be discussed later, each motor is current limited to 2A combining to a total of 6A maximum. Most environmental sensors draw a maximum of a few hundred milliamps, but for safety 2A at 12V is allocated to the sensors. Finally, 2A are allocated to the linear regulators for converting to 5V and 3.3V which are used by the control system and can be diverted to the sensors if necessary.

The 5V and 3.3V linear regulators are rated to provide 3A each. While the control system only requires a few hundred milliamps per rail, the extra current source capabilities may be useful for other sensor configurations. Since the 12V regulator cannot provide the necessary current for 3A regulation on both voltage rails, a secondary battery source can be connected directly to the linear

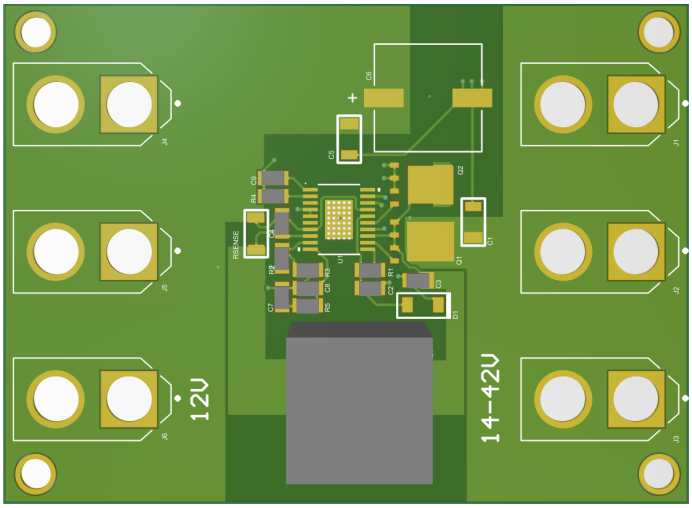

Figure 3.4: 12V DC Buck Converter PCB layout.

regulators instead. For the current configuration of ARES, the full current source capabilities are not required, and therefore the linear regulators are powered from the 12V rail in order to increase efficiency.

3.3.1.1 12V DC Buck Converter

The 12V buck converter is based on the Analog Devices LTC3807, the schematic of which is shown in Figure A.1. A 3D rendering of the board can be seen in Figure 3.4. The buck converter can receive an input voltage from 14.4V to 38V and outputs 12V at up to 10A. At 24V input, the buck converter reaches a maximum efficiency of 96% when outputting 3A and maintains 90%+ efficiency when outputting over 0.1A. The efficiency results in a maximum power loss of 5.5W when at a full load of 120W. To combat overheating, the PCB has the ground pad of the LTC3807 connected directly to the bottom ground plane using vias. The solder mask on the bottom layer beneath the LTC3807 has been removed so that an adhesive heat sink can be attached to provide additional thermal management as necessary.

The board has three input and three output channels, each using an XT-60 connector. While the 60A capability of the XT-60 connector is significantly higher than required, the connector

is standard for high capacity LiPo batteries and is used to conform to the standard. The use of three inputs allows for multiple battery sets to be connected in parallel, increasing the total battery capacity. For this configuration, two 3S 4000$\mathrm{mA\,h}$ LiPo batteries are connected in series resulting in an average output of 22.2V and giving the robot an operational time of 8 hours. In the future, multiple sets of batteries can be used to allow the platform to run for up to a day at a time without recharging. All three 12V outputs are used, one feeds into the 3.3/5V linear regulator, one feeds into the motor driver board, and the third feeds into the CR6 data-logger which in turn provides power to the sensors.

PCB layout required several important design considerations. The signal and power grounds must remain separate, which was accomplished using a modified star-grounding scheme with a ground plane on the same layer as the decoupling capacitors. The sense resistor is measured using Kelvin connections to eliminate unnecessary resistance from traces, and the sense traces are routed side by side with the same path length. The decoupling capacitors are kept close to the LTC3807, reducing path lengths and increasing stability. The switch, trigger, and boost pins are kept as far away from small signals as possible to reduce cross-talk between them. All copper planes are cut out from beneath the inductor to reduce eddy currents and increase efficiency.

3.3.1.2 3.3/5V Linear Regulators

The 3.3/5V power delivery board is designed around the LM108x series of low-dropout positive linear regulators, the schematic of which is shown in Figure A.3. The LM108x series have adjustable regulators able to regulate up to a 29V differential between input and output voltages with a maximum drop out of 1.5V and a current output of 1.5A, 3A, or 5A. For this board, the LM1085 was chosen since it can provide 3A, which is more than is needed but may be useful in the future. When laying out the PCB, the 5V regulator was positioned, allowing it to be cut out and mounted as a stand-alone unit if desired.

The adjustable version of the LM1085 was used, allowing the voltage to be set using a two

resistor voltage divider. The output voltage can be calculated as

$$V_{out} = 1.25V \left(1 + \frac{R_2}{R_1}\right), \tag{3.1}$$

where V_{out} is the regulated output voltage, R_1 is the top resistor of the voltage divider, and R_2 is the bottom resistor of the voltage divider. For voltage input, both an XT-60 connector and a standard 0.1-inch pitch 2-pin header are used. All outputs use the same 2-pin headers. In order to safely conduct at $3\mathrm{A}$, two headers must be used since each header can only handle $2\mathrm{A}$ maximum.

3.3.2 Control System

The control system's goal is to drive the motors based on the desired position and the current position, calculated using encoder measurements. At the center of the control system is a Teensy 4.0, which can be programmed using the Arduino environment, a modified version of C++. The Teensy 4.0 is, at the time of writing, the most powerful micro-controller compatible with the Arduino environment, boasting an ARM Cortex-M7 with an NXP iMXRT1062 chip at a $600\mathrm{MHz}$ clock with a mean current draw of 100mA at 3.3V. The high clock speed allows for a faster control loop resulting in more accurate tracking of the desired path. Important features of the Teensy 4.0 are 3 hardware $\mathrm{I}^2\mathrm{C}$ interfaces with a 4-bit FIFO buffer, 7 serial interfaces with a 4-bit FIFO buffer, and 40 digital General-Purpose Input/Outputs (GPIO).

To receive data about the path the robot should follow, an XBee wifi module was connected to the Teensy 4.0 via a serial connection at 9600 Bd. While a higher baud rate would have been useful, any rate higher than 9600 Bd would exceed the available XBee module's capabilities. In the future, a higher bandwidth XBee wifi module should be used as this will allow two-way communication and more data to be sent to and from the Teensy 4.0 allowing for easier diagnostics of the system as it is running.

To drive the motors, receive feedback from the motors, and receive power, the Teensy 4.0 was

connected to the custom driver board. Motor speeds were sent to the board over an I^2C interface at 400 $\mathrm{KiB\,s^{-1}}$, and the feedback from the motors was connected through the board to the Teensy 4.0's GPIO pins, which were later set as interrupts in order to count the encoder ticks.

3.3.2.1 Motor Driver Board

The motor driver board, shown in Figure 3.5, aims to take an I^2C signal from any $3.3\mathrm{V}$ logic micro-controller, use the signal to control the motors, and return encoder tics to the micro-controller at the required logic level. A schematic of the driver board can be found in Figure A.4. Driving the motors is accomplished using a DRV8871 motor driver capable of driving a brushed DC-motor with up to $2\mathrm{A}$ at voltages from 6.5 to $45\mathrm{V}$. The DRV8871 is controlled using two PWM signals, each driving an h-bridge. The motor current is limited to $2\mathrm{A}$ using a current limiting resistor, the value of which is calculated as

$$I_{lim} = \frac{V_{I,lim}}{R_{I,lim}}, \tag{3.2}$$

where I_{lim} is the maximum current, $V_{I,lim}$ is the limit voltage, and $R_{I,lim}$ is the current limiting resistor value. $V_{I,lim}$ is typically $64\mathrm{kV}$ with a variance of $5\mathrm{kV}$ based on temperature ranging from -40 to $125^{\circ}\mathrm{C}$, but for calculation purposes it is assumed that the value is always $64\mathrm{kV}$. With a desired limit of $2\mathrm{A}$ on the motor driver, the value of $R_{I,lim}$ is calculated to be 32 $\mathrm{k\Omega}$.

In order to reduce the effect from parasitic wire inductance, each motor driver has a 47 $\mu\mathrm{F}$ local bulk capacitor allowing quick current variation and a 0.1 $\mu\mathrm{F}$ IC bypass capacitor, reducing noise near the motor driver.

To save pins on the Teensy 4.0, a PCA9685 is used to generate the PWM signals. The PCA9685 provides 16 12-bit PWM generators allowing for fine control of the motors at a PWM frequency of up to 1526Hz. As previously mentioned, the PCA9685 is driven using an I^2C bus.

The motor encoders run on $5\mathrm{V}$ logic, meaning the signals must be shifted down to 3.3V in

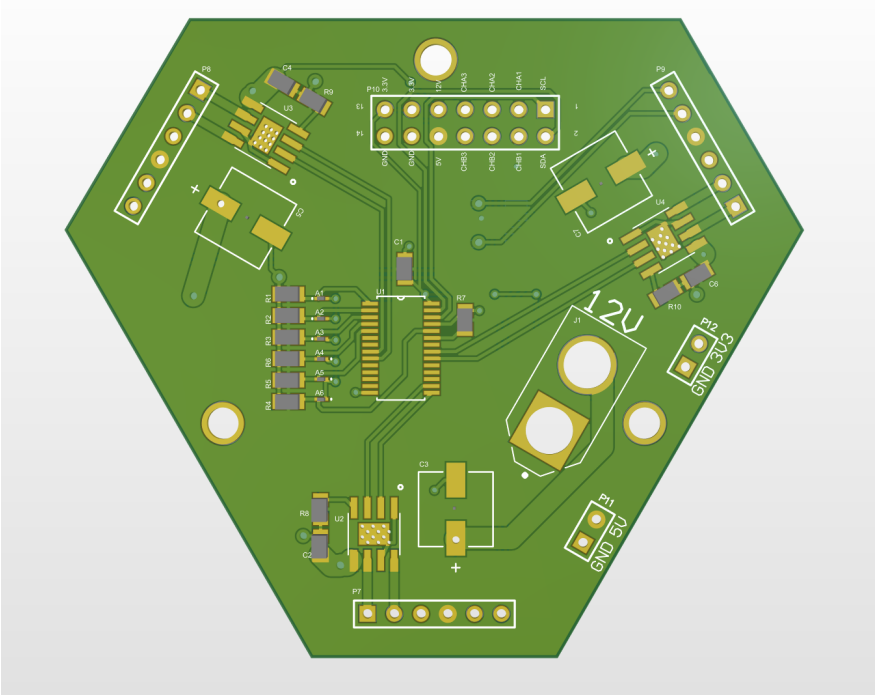

Figure 3.5: Motor driver PCB layout.

order to maintain an acceptable input voltage for the Teensy 4.0. The 5V logic is shifted down to 3.3V logic using a MC14504, a bi-directional six-channel level shifter. Due to the COVID-19 pandemic, there was a shortage of MC14504s, so a CD4504 was used as a drop-in replacement during a revision of the board. The CD4504b is a uni-directional level shifter; however, since the encoders are a uni-directional signal, there are no problems with the replacement.

The board was designed to be placed at the center of the bottom level of the robot. Since all the motors are positioned radially, the board was designed such that the motor headers are also placed radially about the board 120 degrees from one another, providing neat wiring. The PCA9685 was placed in the center of the board, and the level shifter was placed on the bottom of the board. An XT-60 connector was used to provide the 12V rail, while the 5V and 3.3V rails were supplied through 2-pin 0.1-inch pitch headers.

3.4 Sensory Subsystem

For the environmental sensing, a Young81000 ultrasonic anemometer and an HMP60 relative humidity and temperature sensor were used, connected to a Campbell Scientific CR6 data-logger. While the measurement parameters for both sensors and CR6 programming will be discussed in Chapter 5, a brief overview of the sensors and data-logger is discussed here.

3.4.1 Ultra-Sonic Anemometer

The Young81000 anemometer is a three-dimensional ultrasonic anemometer. The ultrasonic anemometer employs three sets of ultrasonic transducers operating at a nominal frequency of 40 kHz. Due to the sensor's geometry interfering with airflow, the anemometer has been calibrated, and look-up tables were implemented to reduce the structure's effect on measured wind velocity as much as possible. Since the anemometer's body partially blocks wind in the z-direction, the maximum measurable wind elevation angle is 60 degrees. The Young81000 anemometer is capable of measuring wind speed with a resolution of 0.01 $\mathrm{m\,s^{-1}}$ and an error of $\pm 1\%\mathrm{rms} \pm 0.05\mathrm{m\,s^{-1}}$. Due to the operating principle of ultrasonic anemometers, an ultrasonic temperature is also measured. In the case of the Young81000 anemometer, the ultrasonic temperature measurement has a resolution of 0.01 K and an accuracy of ± 2 K.

The sensor is mounted on top of the upper rod using an adjustable metal ring clamp about the bottom of the anemometer body. The anemometer is seated on the rod with a mounting depth of 11.5 cm creating a highly stable mount. The sensor was oriented such that it lined up with the robot frame of reference, which will allow for each wind velocity vector to be aligned with the room.

3.4.2 Relative Humidity and Temperature Sensor

The HMP60 relative humidity and temperature sensor uses an INTERCAP® sensor and a 1000 Ω platinum resistance thermometer (PRT) to measure non-condensing relative humidity and temperature, respectively. The sensor measures relative humidity from 0 to 100% with a resolution of $\pm 3\%$ at normal room temperatures with a relative humidity less than 90%. The accuracy decreases to $\pm 5\%$ for relative humidities above 90%. The HMP60 temperature sensor is capable of measuring between -40 and 60°C with an accuracy of ± 0.6 °C, much better than the ultrasonic temperature measurement.

The sensor was mounted in a 41303-5A 6-plate shield, used to protect the sensor from radiation and harsh environments and provide mounting hardware. After securing the HMP60 in the 6-plate shield, the shield was mounted to the top rod far enough below the Young81000 anemometer such that it is unable to interfere with the measured wind velocity.

3.4.3 CR6 Data-logger

A Campbell Scientific CR6 data-logger is used for environmental data collection. The CR6 data-logger is an industry-standard data-logger capable of high-frequency measurements and measurements of sub microvolt analog signals. While the robustness of the data-logger is not necessary for this application, using a CR6 makes adapting the platform to other sensors a simpler task. The data-logger was placed on the second level of the robot. Since the robot undergoes low acceleration, the CR6 data-logger does not need to be secured to the robot and can instead be set on the platform. The CR6 data-logger has little on-board memory, so a 16GB microSD card was used for saving collected data.

Chapter 4

Control Design

In this chapter, non-linear controllers are designed and applied to the platform using kinematic and dynamic models. The controllers' positional accuracy is determined using a visual tracking system to determine which controller is best suited for environmental sensing.

4.1 Kinematic and Dynamic Models

The development of a controller using only the robot's kinematic model may provide acceptable movement accuracy; however, the system performance can be improved by controlling each motor independently. To this end, both the kinematic robot model and the dynamic DC-motor model are developed.

4.1.1 Kinematic Model

The geometry of an omniwheel can be defined using three angles, α, β, and γ, shown in Figure 4.1. Here α is the angle of the wheel's center with respect to the positive x axis in the robot's frame of reference, β is the mounting angle of the wheel with respect to α, and γ is the angle of the rollers with respect to the wheel's axis. Using this geometry, the kinematic model of a three-wheel

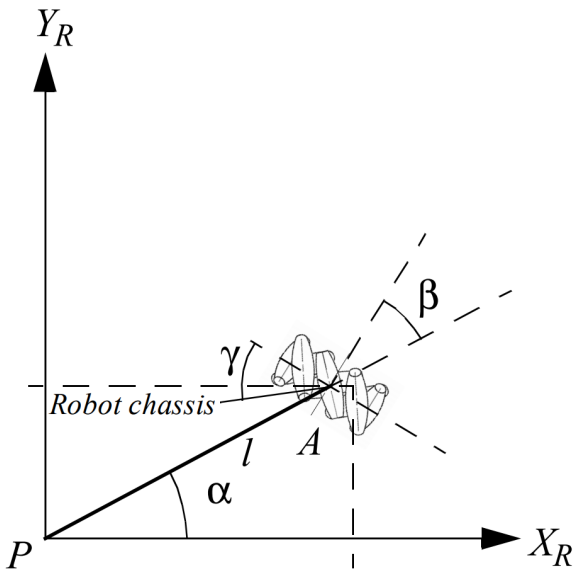

Figure 4.1: Diagram of a Swedish wheel and its corresponding coordinates in the robot's frame of reference (Siegwart et al., 2011).

drive Swedish wheel robot can be derived using the rolling constraint on omniwheels defined by Siegwart et al. (2011) as

$$\begin{bmatrix} \sin(\alpha+\beta+\gamma) \\ -\cos(\alpha+\beta+\gamma) \\ -l\cos(\beta+\gamma) \end{bmatrix}^T \mathbf{R}(\theta)\,\dot{\boldsymbol{\xi}}_I - r\dot{\phi}\cos\gamma = 0, \tag{4.1}$$

where l is the distance from the robot's Center of Rotation (CoR) to the center of the wheel, r is the radius of the wheel, $\dot{\phi}$ is the angular velocity of the wheel, $\dot{\boldsymbol{\xi}}_I$ is the state velocity vector in the global frame defined as $\dot{\boldsymbol{\xi}}_I = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^T$, and $\mathbf{R}(\theta)$ is a rotation matrix described by

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.2}$$

The rolling constraint for the omniwheel only applies when the wheel is not slipping, but slip

can be included by modifying (4.1) as

$$\begin{bmatrix} \sin(\alpha+\beta+\gamma) \\ -\cos(\alpha+\beta+\gamma) \\ -l\cos(\beta+\gamma) \end{bmatrix}^T \mathbf{R}(\theta)\dot{\boldsymbol{\xi}}_I - r\dot{\phi}\cos\gamma = v_s, \tag{4.3}$$

where $v_s = r\dot{\phi} - v_{wheel}$ is the slip velocity of the wheel defined as a function of the wheel velocity v_{wheel}.

With the rolling constraint of a single omniwheel, the robot's geometry can be used to derive ARES' kinematic model. The wheels being used are standard 90-degree Swedish wheels mounted radially about the robot resulting in $\gamma = \beta = 0$. For each wheel i, the rolling constraint simplifies to:

$$\begin{bmatrix} \sin\alpha_i \\ -\cos\alpha_i \\ -l \end{bmatrix}^T \mathbf{R}(\theta)\dot{\boldsymbol{\xi}}_I - r\dot{\phi}_i = v_{s_i}, \tag{4.4}$$

where $\alpha_i = \frac{2\pi(i-1)}{3}$ for $i = 1, 2, 3$. Substituting each α_i into (4.4) results in three equations to describe the robot's kinematics. These equations can be written in matrix form as

$$\mathbf{u} = \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{bmatrix} = \frac{1}{r}\left(\mathbf{B}(\theta)\dot{\boldsymbol{\xi}}_I - \mathbf{v}_s\right), \tag{4.5}$$

where $\mathbf{u}$ is the controllable wheel's angular velocity vector, $\mathbf{v}_s = \begin{bmatrix} v_{s_1} & v_{s_2} & v_{s_3} \end{bmatrix}^T$, and

$$\mathbf{B}(\theta) = \begin{bmatrix} \sin\theta & -\cos\theta & -l \\ \frac{\sqrt{3}}{2}\cos\theta - \frac{1}{2}\sin\theta & \frac{\sqrt{3}}{2}\sin\theta + \frac{1}{2}\cos\theta & -l \\ -\frac{\sqrt{3}}{2}\cos\theta - \frac{1}{2}\sin\theta & -\frac{\sqrt{3}}{2}\sin\theta + \frac{1}{2}\cos\theta & -l \end{bmatrix} \tag{4.6}$$

is a non-linear transformation matrix dependant on the global angular position of the robot.

In order to develop a controller, it is useful to use the inverse kinematic model described by

$$\dot{\boldsymbol{\xi}}_I = \mathbf{B}^{-1}(\theta)(r\mathbf{u} + \mathbf{v}_s) \tag{4.7}$$

4.1.2 Motor Dynamics

The kinematic model is capable of describing the motion of the robot using the angular velocity of each wheel. However, motor controllers are necessary to reach the desired angular velocities for each wheel accurately. To do so, the dynamics of a DC-motor are first derived based on Newton's second law and Kirchhoff's voltage law, respectively, as

$$J\ddot{\theta} + b\dot{\theta} = Ki, \tag{4.8}$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta}, \tag{4.9}$$

where J is the moment of inertia of the motor and all attached hardware, θ is the angular position of the motor shaft, b is the motor's viscous friction, K is the motor constant, i is the current provided to the motor, L is the inductance of the motor, R is the resistance of the motor, and V is the voltage provided to the motor. Writing (4.8) and (4.9) in vector forms results in the motor dynamics and output, respectively, as

$$\frac{d}{dt}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} = \begin{bmatrix}-\frac{b}{J} & \frac{K}{J}\\ -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix}V \tag{4.10}$$

$$\mathbf{z}_{motor} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix}. \tag{4.11}$$

A linear controller can be designed using the state-space model defined by (4.10) and (4.11).

4.2 Controllers

Due to the non-linear nature of the robot's kinematics, non-linear controllers should be designed. In this thesis, a feedback linearization controller and a sliding mode controller were developed. The output from both controllers is the $\mathbf{u}$ vector seen in (4.5); however, the DC-motors are controlled using a PWM signal. Since the PWM signal does not linearly affect the motor's angular velocity, a PID controller is designed for each wheel individually. The control schemes using the feedback linearization controller and the sliding mode controller are shown in Figures 4.2 and 4.3, respectively.

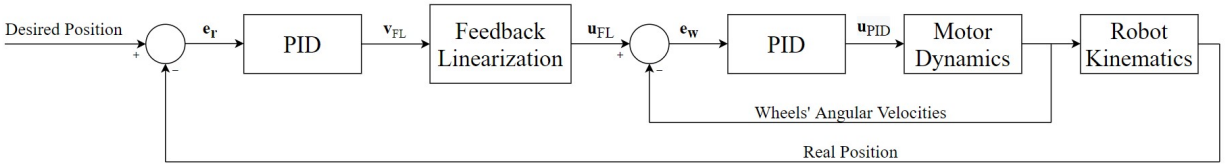

Figure 4.2: Block diagram of feedback linearization controller with a PID controller on each wheel.

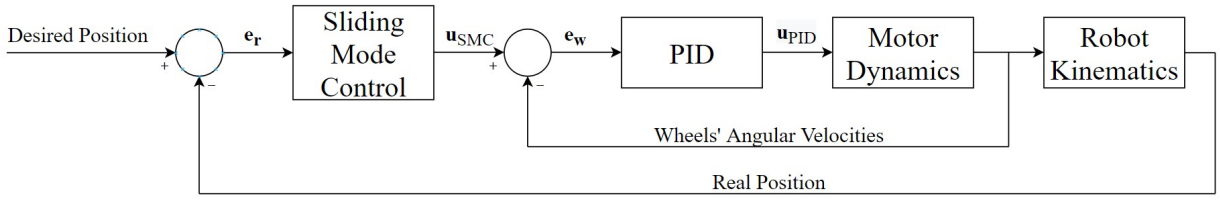

Figure 4.3: Block diagram of sliding mode controller with a PID controller on each wheel.

4.2.1 Feedback linearization

Using the kinematic model of the robot, a feedback linearization controller can be designed. In formulating the feedback linearization in this work a no-slip condition is assumed since ARES does not have the necessary sensors to detect slip in its current configuration. This assumption also simplifies (4.7), which, after converting to error dynamics, is expressed as

$$\dot{\mathbf{e}}_r = r\mathbf{B}^{-1}(\theta)\,\mathbf{u}_{FL}, \tag{4.12}$$

where $\mathbf{e}_r = \boldsymbol{\xi}_d - \boldsymbol{\xi}_I$ and $\boldsymbol{\xi}_d$ is the desired global state of the robot. Linearization is accomplished by defining $\mathbf{u}_{FL}$ using a controller as

$$\mathbf{u}_{FL} = \frac{1}{r}\mathbf{B}(\theta)\,\mathbf{v}_{FL}, \tag{4.13}$$

where $\mathbf{v}_{FL} = \dot{\mathbf{e}}$. The use of $\frac{1}{r}\mathbf{B}$ linearizes the robot kinematics allowing $\mathbf{v}_{FL}$ to be any linear controller.

It is convenient to define $\mathbf{v}_{FL}$ as a Proportional-Integral-Derivative (PID) controller, defined as

$$\mathbf{v}_{FL} = \mathbf{K}_p\mathbf{e}_r + \mathbf{K}_d\dot{\mathbf{e}}_r + \mathbf{K}_i\int_0^t \mathbf{e}_r(t)dt \tag{4.14}$$

where $\mathbf{K}_p$, $\mathbf{K}_d$, and $\mathbf{K}_i$ are the proportional, derivative, and integral 3×3 design matrices, respectively.

Instead of integrating from 0 to t in (4.14), the previous 2 seconds of errors were integrated over reducing excessive overshoot in the error. Through trial and error, the design matrices were

found to be

$$\mathbf{K}_p = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 6 \end{bmatrix} \tag{4.15}$$

$$\mathbf{K}_i = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 30 \end{bmatrix}. \tag{4.16}$$

It was found that the controller performed best when the derivative term was dropped, causing $\mathbf{v}$ to be a PI controller.

4.2.2 Sliding Mode Control

The Sliding Mode Controller (SMC) uses (4.5) and does not exclude the slip term like the feedback linearization controller. The sliding surface is defined as

$$\boldsymbol{\sigma} = \mathbf{e}_r + \boldsymbol{\lambda} \int \mathbf{e}_r dt, \tag{4.17}$$

where $\boldsymbol{\sigma}$ is the sliding surface, $\boldsymbol{\lambda} \in \mathbb{R}^{3x3}$ is a positive design matrix, and $\mathbf{e}_r = \boldsymbol{\xi}_d - \boldsymbol{\xi}_I$. The equivalent control portion of the input is defined by setting the derivative of the sliding surface to zero and solving for the control vector $\hat{\mathbf{u}}_{SMC}$ as

$$\dot{\boldsymbol{\sigma}} = \dot{\mathbf{e}}_r + \boldsymbol{\lambda}\mathbf{e}_r = \mathbf{B}(\theta)^{-1}(r\hat{\mathbf{u}}_{SMC} + \hat{\mathbf{v}}_s) - \dot{\boldsymbol{\xi}}_d + \boldsymbol{\lambda}\mathbf{e}_r = 0 \tag{4.18}$$

$$\hat{\mathbf{u}}_{SMC} = \frac{1}{r}\left(\mathbf{B}(\theta)\left(-\boldsymbol{\lambda}\mathbf{e}_r + \dot{\boldsymbol{\xi}}_d\right) - \hat{\mathbf{v}}_s\right) \tag{4.19}$$

where $\hat{\mathbf{v}}_s$ is a vector containing the estimated slip of each wheel. Since ARES does not have any sensors for determining slip, it is assumed that $\hat{\mathbf{v}}_s = \mathbf{0}$ and the error created by this is handled by

the SMC.

In order to maintain the system on the sliding surface, the discontinuous portion of the controller is defined as

$$\mathbf{u}_{c,SMC} = -\mathbf{B}(\theta)\,\mathrm{ksign}(\boldsymbol{\sigma}) = -\mathbf{B}(\theta)\,\mathrm{ksign}\left(\mathbf{e}_r + \boldsymbol{\lambda}\int \mathbf{e}_r dt\right), \tag{4.20}$$

where $\mathbf{k} \in \mathbb{R}^{3x3}$ is a positive definite design matrix. Combining (4.19) and (4.20), the complete controller is defined as

$$\begin{aligned}\mathbf{u}_{SMC} &= \hat{\mathbf{u}}_{SMC} + \mathbf{u}_{c,SMC} \\ &= \frac{1}{r}\left(\mathbf{B}(\theta)\left(-\boldsymbol{\lambda}\mathbf{e}_r + \dot{\boldsymbol{\xi}}_d\right) - \hat{\mathbf{v}}_s\right) - \mathbf{B}(\theta)\,\mathrm{ksign}\left(\mathbf{e}_r + \boldsymbol{\lambda}\int \mathbf{e}_r dt\right).\end{aligned} \tag{4.21}$$

Considering the Lyapunov function $V = \frac{1}{2}\boldsymbol{\sigma}^2$, the constraints on $\mathbf{k}$ can be derived to guarantee stability as

$$\begin{aligned}\frac{dV}{dt} &\leq \eta\,|\boldsymbol{\sigma}| \\ \boldsymbol{\sigma}^T\dot{\boldsymbol{\sigma}} &\leq \eta\,|\boldsymbol{\sigma}| \\ \boldsymbol{\sigma}^T\left[\mathbf{B}(\theta)^{-1}(\mathbf{v}_s - \hat{\mathbf{v}}_s) - \mathrm{ksign}(\boldsymbol{\sigma})\right] &\leq \eta\,|\boldsymbol{\sigma}| \\ \eta + \mathbf{B}(\theta)^{-1}(\mathbf{v}_s - \hat{\mathbf{v}}_s) &\leq \mathbf{k}\end{aligned} \tag{4.22}$$

where η is a small positive constant. From this derivation it can be seen that $\boldsymbol{\sigma}^T\mathbf{k} = \mathbf{k}\boldsymbol{\sigma}^T$ which is achieved by making $\mathbf{k}$ a symmetric matrix.

The $\boldsymbol{\lambda}$ and $\mathbf{k}$ design matrices were roughly tuned in simulation and more finely turned during

testing. The design matrices are

$$\boldsymbol{\lambda} = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 6 \end{bmatrix} \tag{4.23}$$

$$\mathbf{k} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}. \tag{4.24}$$

4.2.3 PID Controller for Motors

The kinematic model of the robot uses the velocity of each wheel as the system input. As such, all the controllers developed for the robot output a control signal corresponding to the desired angular velocity of the wheels. To achieve accurate angular velocity in a reasonable time frame, a PI controller was implemented on each motor. The derivative term was omitted since testing showed the controller performed best without it. Since the control input to the motors is voltage, the design matrices in (2.5) are reduced to scalar gains. Through modelling and testing, the appropriate gains were found to be $K_{p_w} = 10$ and $K_{i_w} = 30$. Similar to the feedback linearization controller, the integral of error was calculated over the previous 2 seconds in order to reduce oscillations caused by the integral term.

4.3 Physical Parameters

A number of physical parameters are required in order to control the robot accurately. It is important to obtain accurate values to ensure good performance. All necessary parameters are listed in Table 4.1.

Parameter	Symbol	Value
Motor Viscous Friction	b	$2.6\ \mathrm{N\,m\,s}$
Axle Moment of Inertia	J	$2.13 \times 10^{-5}\ \mathrm{kg\,m^2}$
Motor Constant	K	$105.21\ \mathrm{N\,m\,A^{-1}}$
Armature Resistance	R	$10.4\ \Omega$
Motor Inductance	L	517 µH
Wheel Radius	r	$5.08\ \mathrm{cm}$
Wheel to CoR Distance	l	$26.32\ \mathrm{cm}$

Table 4.1: Parameters required for the motor dynamic and omniwheel kinematic models.

The wheel radius and distance between each wheel and the robot's center of rotation were determined using CAD models. The axle's moment of inertia was calculated using the mass and radius of the axle, wheel, and wheel hubs. The motor torque constant and motor viscous friction are listed in the motor's datasheet. The armature resistance and motor inductance were found experimentally using a Maxwell-Wien bridge and an arbitrary signal generator.

4.4 Controller Verification

Having developed the feedback linearization and sliding mode controllers, experiments were conducted to determine which controller provided the most reliable and accurate performance. An accurate tracking system was required to quantify the controller performances without reliance on positional data collected by the robot. This tracking is accomplished through the use of a VICON system. The VICON system consists of a set of infra-red cameras that can track Infra-Red (IR) reflectors. By knowing each camera's location and combining the video feeds from the cameras, the locations of each marker can be calculated with a resolution of 0.1mm and an error of ± 0.2mm. The VICON system data can be streamed in real-time to a MATLAB environment, where data about each reflector position was collected and analyzed.

The robot frame had eight reflectors attached to it. Six of the reflectors were positioned at each outer corner of the frame. The last two reflectors were placed near the first wheel of the robot,

breaking the reflectors' symmetry and allowing for the robot's angular position to be calculated. Using the robot's radius and three reflectors, the center of the robot was calculated, allowing for the x and y positions of the robot to be obtained. By averaging the position of the two points closest to the first wheel and subtracting the x and y position of the robot, the robot's angular position was calculated. Code used during data collection is presented in Appendix B.

The desired robot trajectory was sent over WiFi using the xBee module. The script that collected positional data from the VICON system also contains the robot's desired path such that the path could be sent at 50Hz, the same frequency as the controller. The path was calculated such that it would not exceed the maximum speed of the robot. Originally, the system was set to run at its maximum of 100Hz; however, the xBee module is only capable of 9600 Bd, limiting the data throughput to the robot, so the VICON frequency was reduced to 50Hz. The low baud rate also means the robot cannot send its calculated position back to the user at 50Hz; therefore, only the desired and real robot positions are tracked. While these parameters are all that are needed for verifying the controllers, the robot's calculated position would have aided in the tuning of the controllers and troubleshooting.

Fine-tuning of the controllers was accomplished through trial and error over multiple test paths. Once the controllers had been tuned to produce acceptable margins of error, the robot was sent on two zig-zag pattern paths in order to calculate and compensate for drift. Once drift compensation had been accomplished, other paths were executed to quantify each controller's performance.

4.4.1 Drift Compensation

The robot was set on a path that had it sweep the room by travelling to a set of equally spaced positions, shown in Figure 4.4. The robot was given 8 seconds to reach each position. Bias was calculated for each test in order to determine drift per meter. The side-to-side motion of the path allows for bias to cancel itself; therefore, the robot was rotated 90 degrees, and the tests were run

a second time to determine drift for both the x and y directions. Each test was repeated 10 times in order to reduce random error.

Once testing had been completed and bias calculated, the drift was used to adjust the robot's position calculation, and the tests were re-run to determine if the drift was sufficiently reduced. Since the drift is caused by small errors in physical parameters and encoder measurements, either controller could be used for the tests. The SMC was chosen due to preliminary tests indicating that it had lower random error than the feedback linearization controller.

As shown in Figure 4.4, the robot tended to overshoot when travelling in the positive x direction. Figure 4.5 plots the error between the robot's real and desired positions as a function of time. Using a least-squares method, the line of best fit is plotted, providing drift as a function of time. A simple conversion can be performed to show that drift in the x direction is 0.0199 $\mathrm{m\,m^{-1}}$.

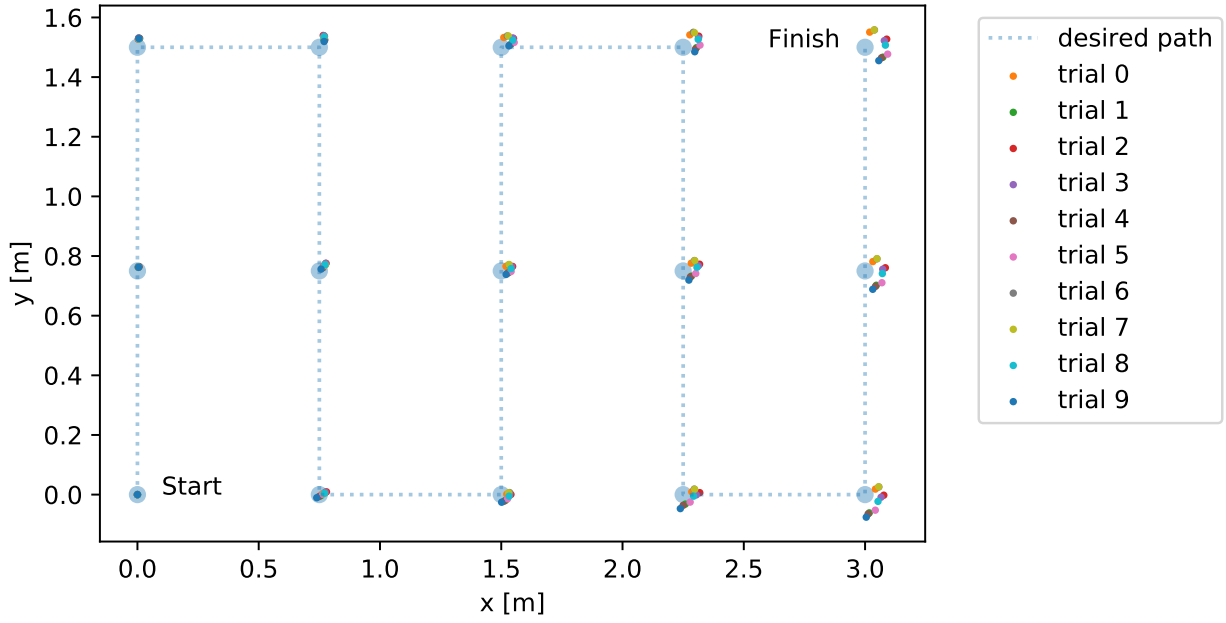

Figure 4.4: Position of the robot at each test point over the path moving in the positive x direction.

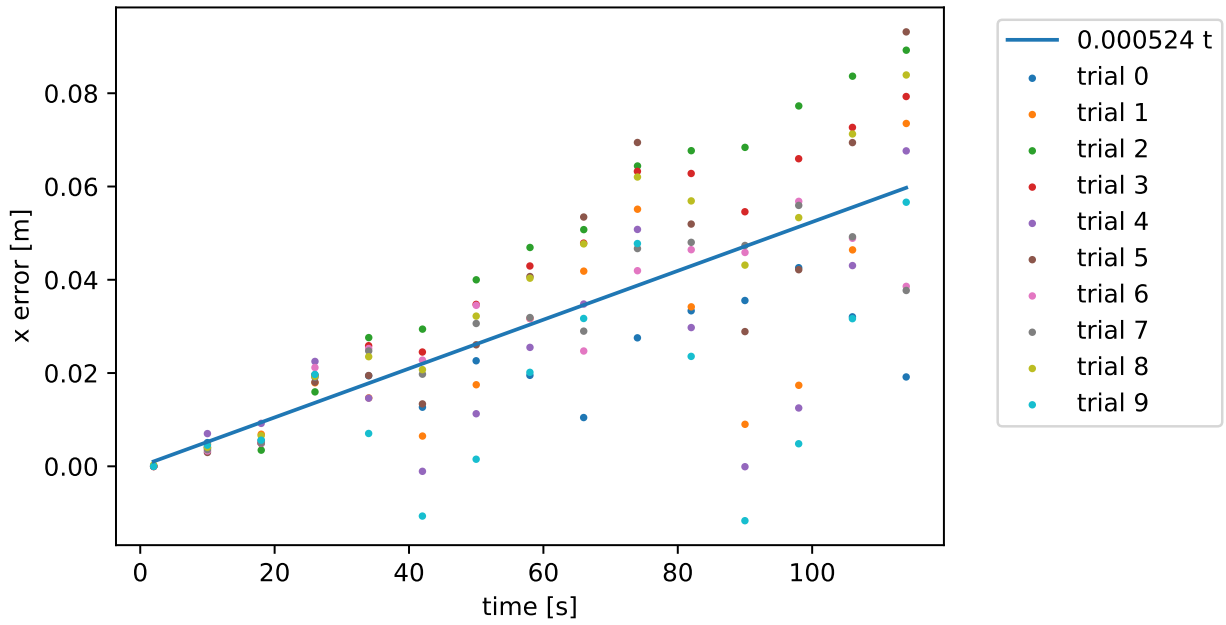

Figure 4.5: Error as a function of time, linear fit shows positional error as a function time corresponding to a drift of 0.0199 $\mathrm{m\,m^{-1}}$.

Running the same tests for the y direction provides similar results, as seen in Figure 4.6. The robot tends to overshoot the y position, which is quantified in Figure 4.6 using a line of best fit. Converting to drift, the robot exhibits a drift of $0.0259\ \mathrm{m\,m^{-1}}$ in the y direction, which is comparable to the drift in the x direction. The drifts' similarity confirms that modelling error is responsible for drift and can be remedied by scaling the calculated position.

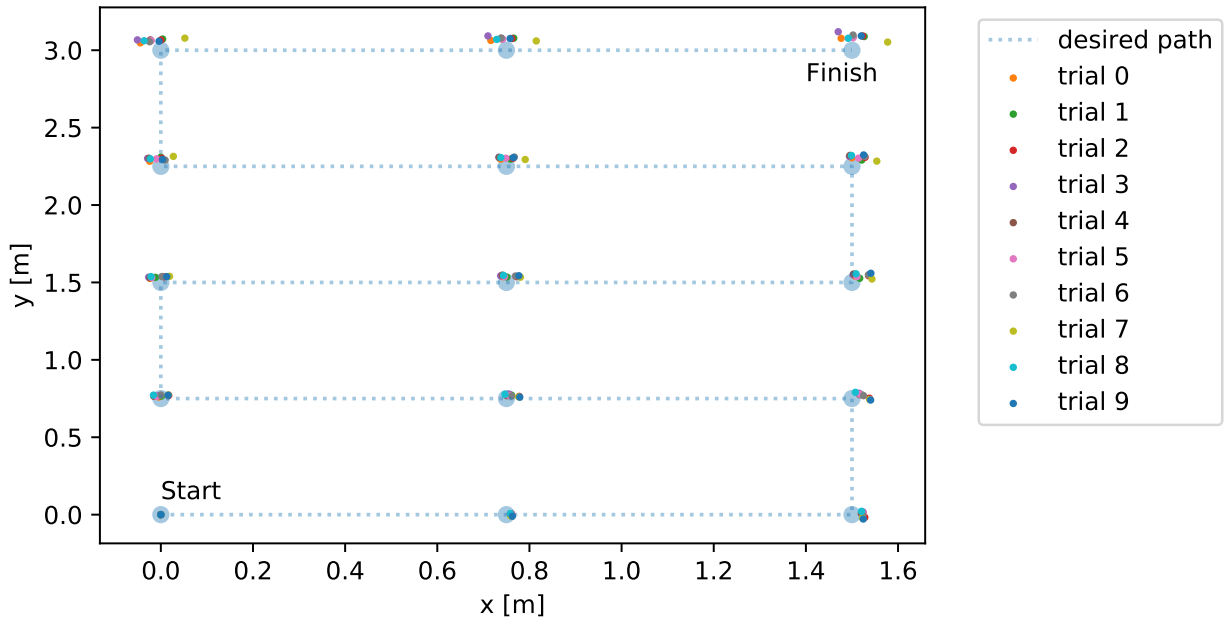

Figure 4.6: Position of the robot at each test point over the path moving in the positive y direction.

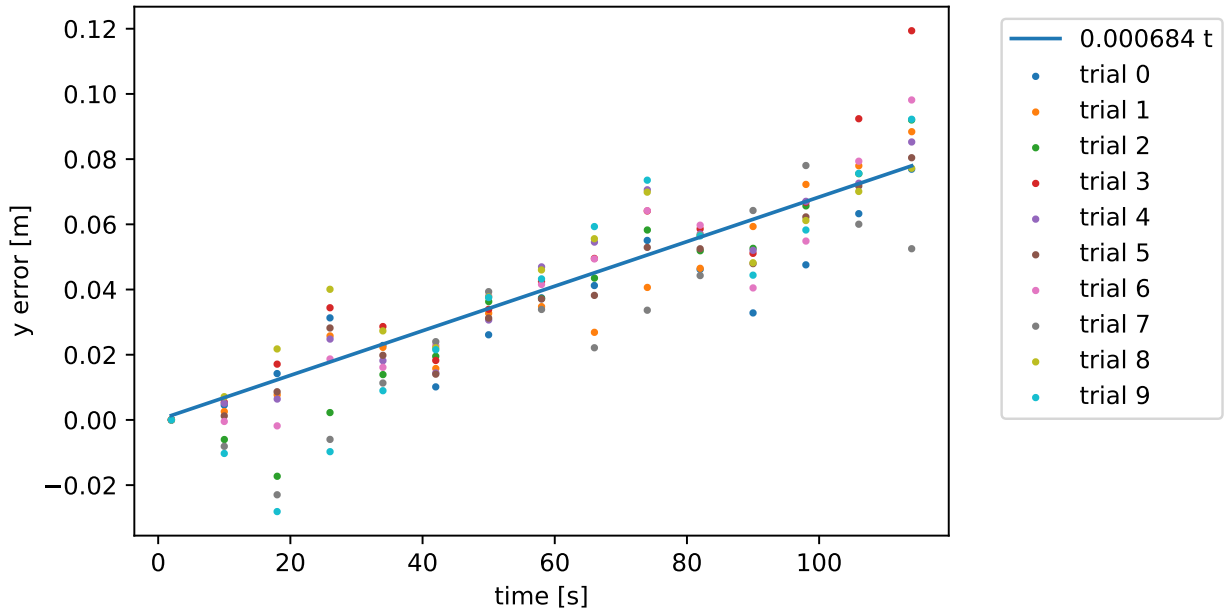

Figure 4.7: Error as a function of time, linear fit shows positional error as a function time corresponding to a drift of 0.0259 $\mathrm{m\,m^{-1}}$.

Using this data, the position calculation was scaled in order to reduce the bias. Both tests were re-run 5 additional times with drift compensation and plotted in Figures 4.8 and 4.9. Drift was calculated using the same method as before and showed a marked improvement with drifts of 2.9 $\mathrm{mm\,m^{-1}}$ and 2.0 $\mathrm{mm\,m^{-1}}$ in the x and y directions, respectively.

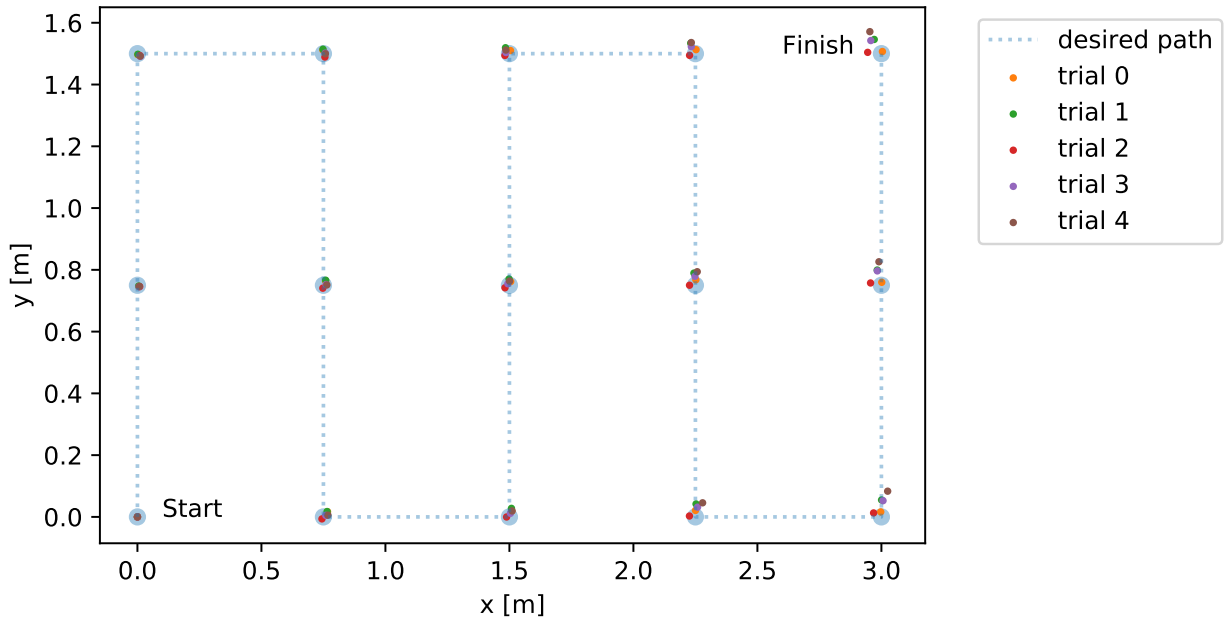

Figure 4.8: Position of the robot at each test point over the path moving in the positive x direction after drift compensation.

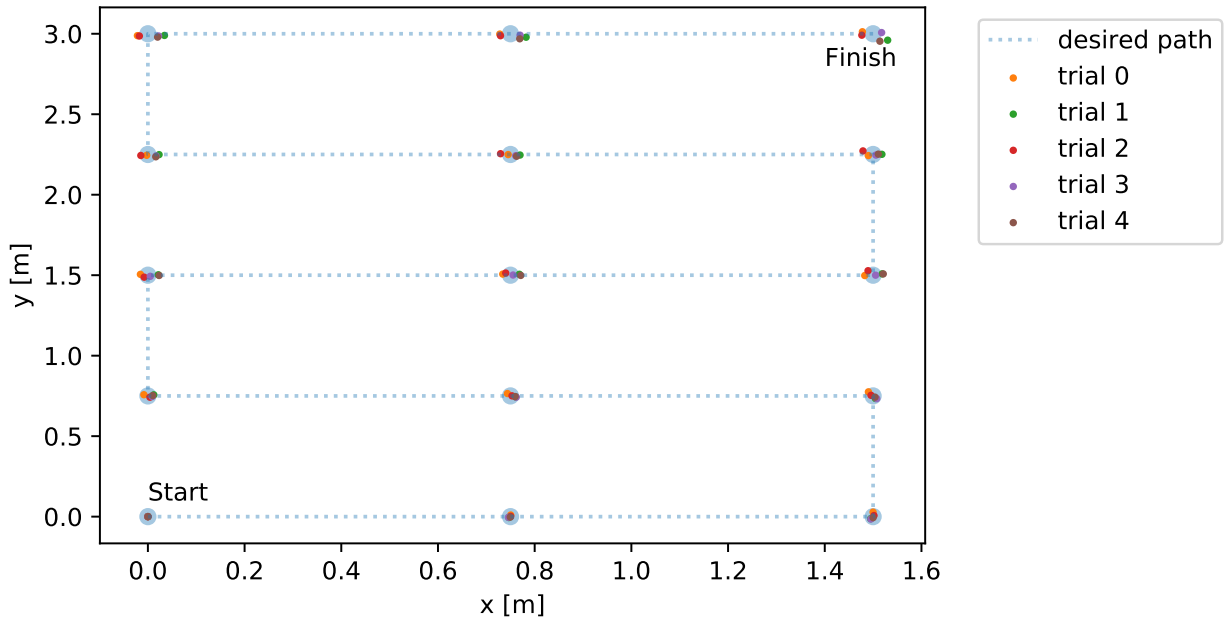

Figure 4.9: Position of the robot at each test point over the path moving in the positive y direction after drift compensation.

One of the major issues with accurately controlling the robot is its inaccurate prediction of angular position. The robot's motors and wheels have small tolerances, allowing the wheels to rotate slightly without rotating the primary motor shaft, from which the encoders take their measurements from. This allows for the robot's real angular position to be up to a degree off from the calculated angular position. While this is a fairly small error when applied to environmental sensing, over the course of 1.5 m an angular error of 1 degree corresponds to a drift of 2.6 cm. In addition, each time the robot stops, the angular error has a chance to increase as the wheels have reset their position. While this compounding over a large set of starts and stops should result in a net 0-degree error, it is possible to locally have it compound to a few degrees resulting in the larger random error. In the future, encoder data should be fused with data from a gyroscope to determine angular position more accurately.

With the drift compensated for, both controllers' performance can be quantified using different

paths, starting with a parametric rose trajectory.

4.4.2 Rose Trajectory

Both controllers were tested on a 4-petal rose trajectory, shown in Figure 4.10, over the course of 120 seconds, covering a total distance of 19.37 meters. The rose plot's symmetry reduces error due to bias and results in random error dominating the total measured error. The rose trajectory is described as

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\pi t}{15}\right)\cos\left(\frac{\pi t}{30}\right) \\ \cos\left(\frac{\pi t}{15}\right)\sin\left(\frac{\pi t}{30}\right) \\ 0 \end{bmatrix}. \tag{4.25}$$

Each test was run 5 times, and the results were averaged to reduce bias from a single run. The time-independent Root Mean Squared Error (RMSE) of each controller is shown in Table 4.2. Graphs of the robot position when using the feedback linearization controller and the SMC can be seen in Figures 4.10 and 4.11, respectively. As can be seen, the SMC outperforms the feedback linearization controller by a factor of four, having an average translational RMSE of 4.2 mm and a rotational RMSE of 0.0058 rads.

Table 4.2: Time independent RMSE for each dimension during parametric rose trajectory.

Controller	x RMSE (mm)	y RMSE (mm)	θ RMSE (rad)
Feedback Linearization	18.4	20.3	0.016
Sliding Mode Control	4.1	4.3	0.0058

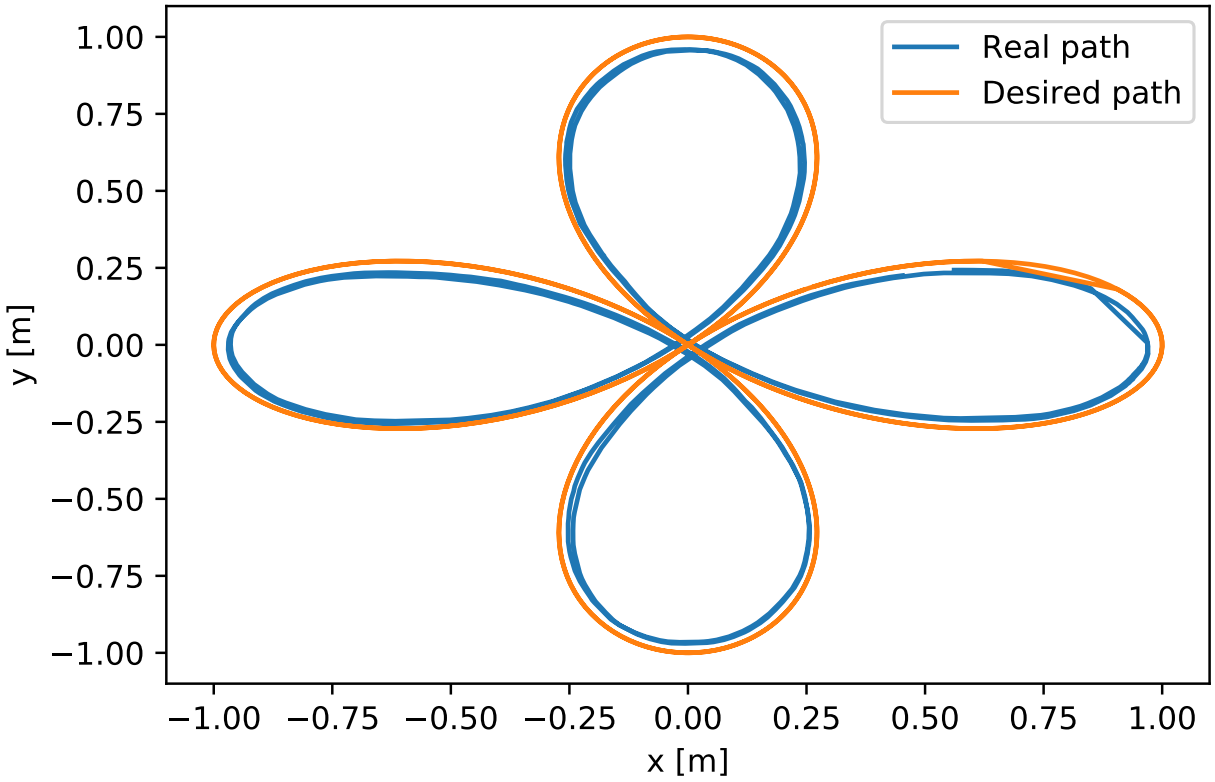

Figure 4.10: Omniwheel robot path compared to desired path using Feedback Linearization.

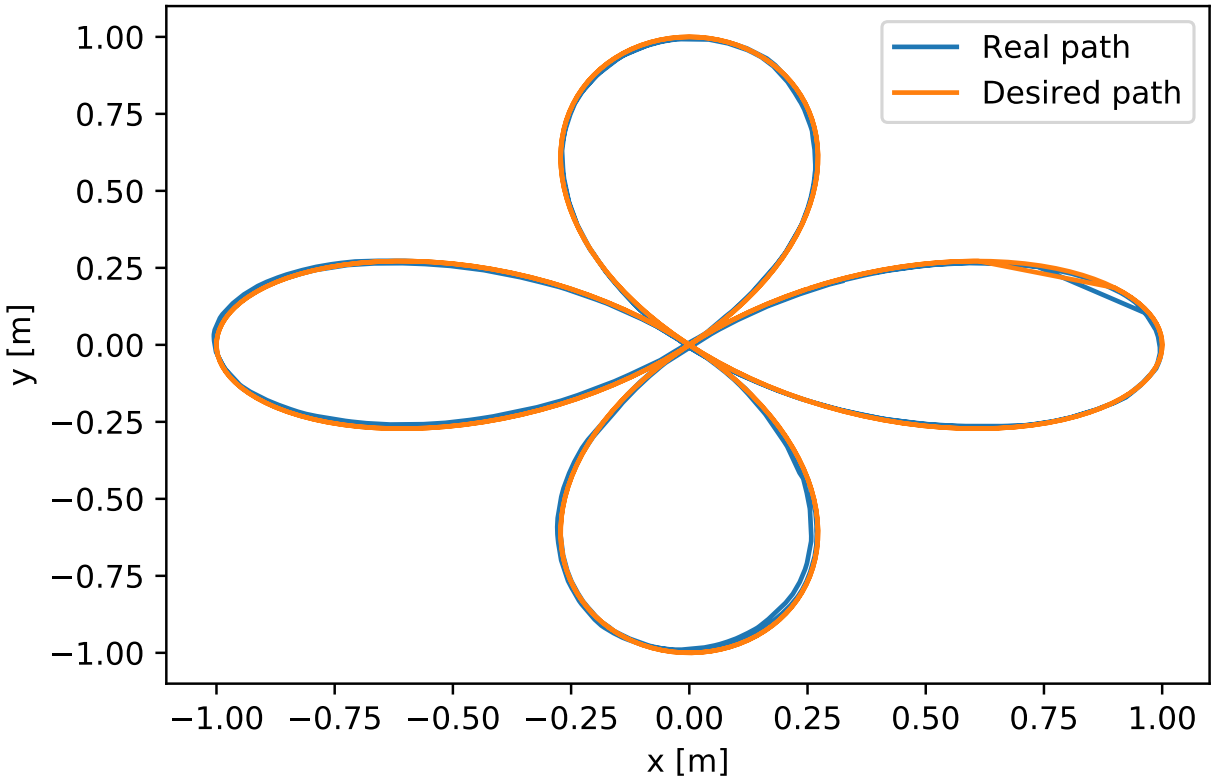

Figure 4.11: Omniwheel robot path compared to desired path using SMC.

4.4.3 Random points

While the robot will never rotate during the environmental experiments, it is useful to quantify how well the controllers perform when the robot is rotating and translating simultaneously. Therefore, the controllers are tested by moving the robot to random positions in the room, testing the controllers' ability to stabilize from any initial state. For each test, 11 positions were randomly generated, and the robot was given 20 seconds to reach each position before being sent the next one.

Position error for the feedback linearization controller is shown in Figure 4.12. The feedback linearization controller performs reasonably when reducing error in the x and y directions; however, it has difficulties reducing error in angular position. Additionally, the use of a PID controller

results in a long settling time.

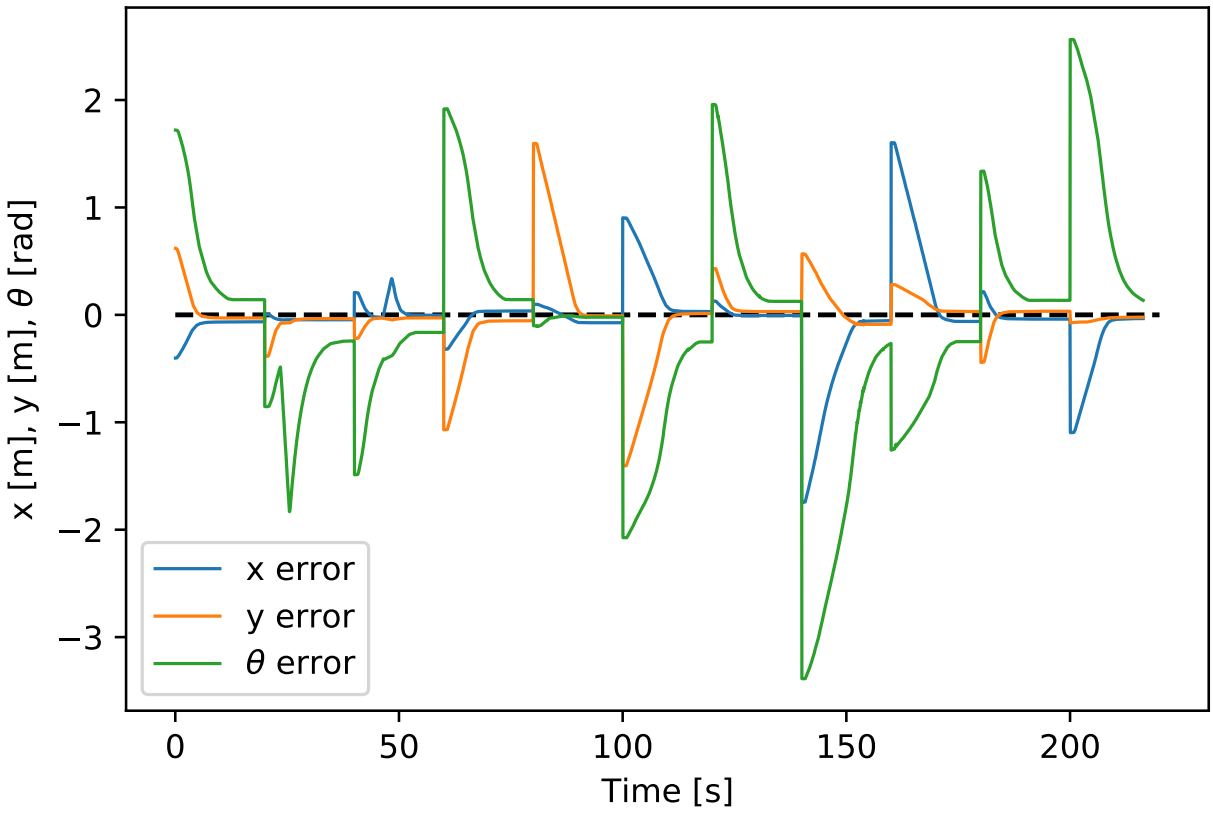

Figure 4.12: Feedback linearization controller error in translational and rotational positions as a function of time when stabilizing to random positions.

The SMC significantly outperforms the feedback linearization controller both in convergence time and error reduction, as seen in Figure 4.13. Other than small deviations, the SMC converges to zero error in a linear fashion and exhibits no overshoot, making it ideal for environmental sensing, where time spent travelling from one position to another is time lost for taking environmental measurements. The sliding mode controller also has no issues with achieving the desired angular position thanks to the discontinuous control signal forcing the robot onto the sliding surface.

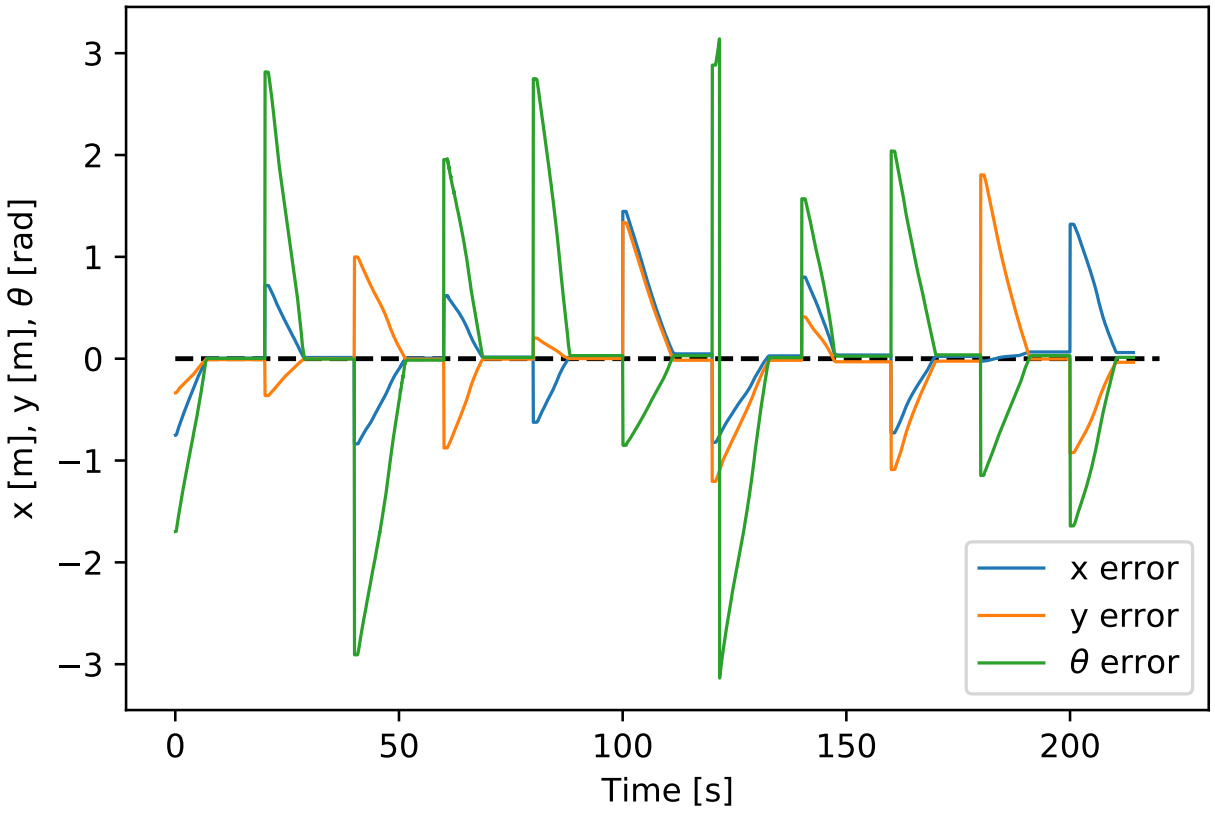

Figure 4.13: Sliding mode controller error in position as a function of time when stabilizing to random positions.

These results show that the sliding mode controller performs significantly better than the feedback linearization controller. Therefore the sliding mode controller will be used for all environmental sensing experiments.

Chapter 5

Environmental Analysis

To verify ARES as a viable platform for indoor environmental sensing, the robot was tested in a lab environment. In this chapter, the specifics of the room in which ARES was tested are discussed, the platform's operation is explained, and a D* Lite path planning algorithm is used to generate collision-free paths for ARES to follow. After processing the collected environmental data, environmental statistics in the room are discussed, and thermal comfort predictions are made.

5.1 Experimental Set-up

The environmental testing needed to be executed in an indoor office environment. Two options were readily available, the VICON lab and the Mechatronics lab. The VICON lab would allow for easy tracking of the robot position; however, the floor space is extremely small with few obstacles, which would likely lead to fairly un-interesting data to be collected. Additionally, the VICON lab is used to teach undergraduate courses and is not available for full 24 hour periods at a time. The Mechatronics lab is a well-used workspace with multiple large workstations. Figure 5.1 shows a simplified CAD model of the room, with all important features. The size of the Mechatronics lab allows for the robot to take measurements spatially far apart, both near work stations and along

common walking spaces. Without the VICON system for positional tracking, the robot must rely on encoder measurements to determine its position. As shown in Chapter 4, the encoders provide sufficient positional accuracy when using the sliding mode controller. Since the robot does not always start in the exact same spot, and more importantly, with a slight rotation, drift can easily accumulate. Therefore, when running experiments, the experimenter needed to be in the room to make sure it reached the first set point accurately. When the robot did not accurately reach the set point, it was manually adjusted slightly to make sure it is lined up properly for the rest of the run. In the future, implementation of a Simultaneous Localization and Mapping (SLAM) system would allow the robot to take measurements fully autonomously.

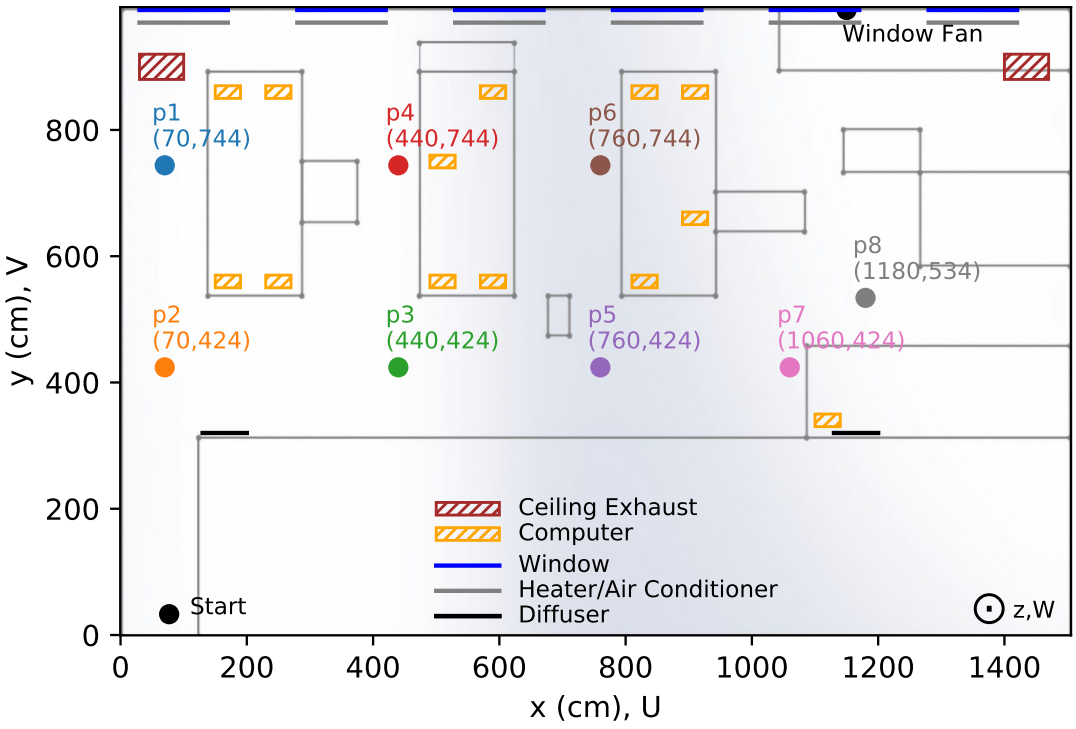

Figure 5.1: Map of the Mechatronics lab with robot start location, positions to measure the environment, and the position of the window fan.

At the time of the experiments, the HVAC system was set to cool the room down. This was accomplished using an air conditioner below each window on the outer wall of the room, which pushed cooled air up towards the ceiling. There is a bulkhead with two vents acting as diffusers on the opposite side of the room. Since the room is used as an office space, many powerful computers are often running all day. Each computer is positioned under the desk on a small elevated platform, allowing the computer's cooling system to more easily eject hot air out the bottom of the computer case. The number of computers in the room causes the room to be uncomfortably warm, as such a fan is placed in front of a slightly open window. The fan is run continuously, allowing the room to cool down overnight, so the room is at a comfortable temperature in the morning.

Measurement points in the room were chosen based both on spacing and frequency of use. Positions 1, 4, 6, and 7 were chosen due to their proximity to workstations. These points are positioned between workstations to obtain a reasonable measurement of the environment where people would normally be sitting. It should be noted that position 1 is within one meter of the workstation that the experimenter sat at, as such data at this point will be influenced based on what the experimenter was doing at the time. Due to the COVID-19 pandemic, nobody else was in the lab, but the lab computers were still being used remotely since all tests were conducted on weekdays. When evaluating thermal comfort, these positions will be considered as places that people normally work while sitting.

Positions 3 and 5 are located near the whiteboard, where people tend to stand writing on the board or walk along to reach their workstations. Position 2 is located at the entrance to the room where there is high traffic, especially when the room is used for undergraduate labs. Position 8 is positioned near a 3-D printer, laser cutter, and the area where undergraduate labs are conducted. Position 8 is also in the airflow path from the fan sitting in the window, causing more draftiness than other areas of the lab. These areas are most commonly used while standing or walking; thus, these working conditions will be considered when evaluating thermal comfort.

5.1.1 Datalogger and Instrument Set-up

In order to collect data from the Young 81000 ultrasonic anemometer and the HMP60 temperature and relative humidity sensor, the CR6 data-logger had to be correctly programmed. The CR6 data-logger was set up with differential voltage inputs for each data point collected (e.g. U component of wind velocity, relative humidity, etc.). Both the ultrasonic anemometer and temperature and relative humidity sensors use an output voltage of 5V to encode their data. By dividing each measurement range by its output voltage range, conversion equations for voltage to wind velocity components, relative humidity, or temperature were found. These conversions were implemented in code, allowing the CR6 data-logger to save data using units of meters per second for wind velocity components, percentages for relative humidity, and Kelvin for temperature. Wiring diagrams for the sensors and CR6 data-logger can be found in Appendix C.2.

The data was collected at 10Hz and saved to a microSD card every fifteen minutes. While the CR6 data-logger stores a long-term file with all data collected since the last reset, this file overwrites itself as more data is collected and sometimes acts spuriously. To avoid data loss, a subfunction was called to save each minute of data as its own file. The data was written to the microSD card every 15 minutes in one-minute increments but could be forced to save using the eject button. When the eject button is pressed, the CR6 data-logger enters a saving period during which the "act" LED turns red. One must wait until the "act" LED switches to green before removing the microSD card. Removing the SD card before it has been properly ejected may lead to loss of data, so the card was always removed over one minute after data had been collected, and special care was taken not to power off the robot while files were being saved.

The code was uploaded to the CR6 data-logger using the logger-net application provided by Campbell Scientific. If the logger-net application is not available due to the software trial period expiring, the PC200 software can be used to upload code and download data instead. However, the software does not provide documentation for the proprietary language Campbell Scientific

dataloggers use. When programming the CR6 data-logger, the PC200 application was often used to pull data from the CR6 data-logger without saving it to the SD card to confirm that data was being properly collected. The code used during experiments is included in Appendix C.3.3.

5.1.2 Path Planning

The robot's path was generated using a D* Lite algorithm modified from a python version written by user zhm-real on GitHub[1]. The room was defined using the simplified CAD drawing of the room and a pixel resolution of 1 cm. To ensure the robot would not run into any obstacles, the radius of each obstacle was increased by 0.4 meters, slightly larger than the robot's radius. Paths were generated between positions by feeding the positions into the D* Lite algorithm as start and end points sequentially (i.e. start = position 1, end = position 2). These paths were stitched together, resulting in a full half-hour path. It should be noted that the robot pushes lightly against the door at the start of each run before continuing to position 1. Doing so helps align the robot with the room by causing it to rotate slightly in order to be parallel to the door, helping alleviate some of the previously mentioned alignment issues.

The D* Lite algorithm was chosen in part due to its ability to adapt to different environments. In this case, the environment is not changing; however, using the D* Lite algorithm allows for imaginary obstacles to be placed to make small manual adjustments to the path easily. This was particularly useful on the path between positions 6 and 7, where the robot originally passed extremely close to a desk when taking the shortest path. To prevent this, the desk polygon was expanded slightly using imaginary obstacles to keep the robot at least 7 cm from any obstacles. The imaginary obstacles were also used to smooth out the paths such that there were fewer sudden changes in momentum, reducing slip and increasing the robot's positional accuracy.

A script was written to take the D* Lite algorithm paths and prepare them for use on ARES. Since the robot does not have enough on-board memory to store the path, the path was sent to

[1] https://github.com/zhm-real/PathPlanning

the robot at 50Hz, the same frequency as the controller. The robot would receive a position to move to, attempt to reach it and then receive the next position along the trajectory at its next time step. For ARES to maintain its trajectory accurately, the speed required to maintain the trajectory cannot exceed the robot's maximum speed. Therefore the script created intermediate steps in the generated path such that when stepped through at 50Hz, the maximum speed of the robot would not exceed 0.3 $\mathrm{m\,s^{-1}}$. When the robot reached one of the set positions, the script inserted the same point multiple times, causing the robot to wait at the position for a set amount of time. In all tests, the robot waited for 160 s at each position to take measurements. The result is a total trajectory execution time of 24.6 min giving approximately five minutes for batteries to be changed and data to be collected between tests.

5.1.3 Experiment

Testing was carried out over two periods. The first period lasted from 1530 EDT on October 2, 2020, until 1000 on October 3, 2020, while the second period was executed on October 8, 2020, from 0800 to 1530. Originally the experiment was expected to be run a full 24 hours from October 2nd to October 3rd; however, the experimenter could not stay awake to reset the robot at 1030 on the 3rd, so the last eight hours of testing was moved to a separate day. October 8th was chosen both due to the outdoor environment being similar to the previous testing period and the need to wait for a COVID test to return. The second set of testing began at 0800 to properly capture the four-hour period starting at 0800. This will become more important later as the increase in temperature starts at approximately 0730, allowing the data starting from 0800 to capture the changing environment more accurately.

Tests were executed at the start of each hour and half-hour, creating a data set for the environment. One minute after the robot had finished its run, it was placed back at the start point in preparation for the next run. One of the two batteries was changed and charged on the hour, re-

sulting in each battery running for two hours before being re-charged. While the batteries should last over eight hours of run time in theory, the batteries were changed often to avoid any chance of over-drain since the batteries are LiPos and known to be damaged when over-drained. Every four hours, the data was collected from the CR6 data-logger and checked to ensure no data was missing.

It should be noted that for the run starting at 1930, the robot was interfered with by a moved chair. This caused ARES to fail to reach positions 3 and 4; therefore, these positions are excluded from data analysis.

5.2 Data Processing

Before the results can be analyzed, the data must be cleaned and processed. Since the positional data is collected locally on the computer and the environmental data is collected using the CR6 data-logger, the data sets are combined using the timestamps as indices. While the positional data is guaranteed to be collected starting exactly on the second, the CR6 data-logger data can start at any time, so the nearest earlier time in the positional data is combined with the CR6 data-logger data. Once all 24 hours of data had been combined, it is split into half-hour windows corresponding with each run. Each half-hour window is then re-indexed by position and saved in a file named using the start time of the run (i.e. 08-30). It should be noted that the data collected was removed while the robot was travelling between measurement locations. The first and last 5 seconds of data at each point are removed to remain certain that there is no data while moving included in the analysis, and the effect from robot movements on the environment is minimized.

A second python script was written for data analysis. The script first read all the data into pandas MultiIndex data-frames, allowing the data to be indexed both by the start time of the run and the position where the data was collected. Data for each position is used to calculate the average wind velocity components in the x, y, and z directions along with the average total wind speed, the

average relative humidity, and the average temperature as read by the ultrasonic anemometer and the HMP60 sensor. The ultrasonic temperature data was plotted against the HMP60 temperature data to create a calibration curve. This adjustment was applied to the ultrasonic temperatures and used for the rest of the analysis. The ultrasonic temperature was used instead of the HMP60 temperature data since the HMP60 sensor has a large thermal mass, causing a slower response to temperature changes. With the short measurement period at each point, the temperature sensor does not have time to fully reach equilibrium resulting in inaccurate readings. For this reason, the calibrated ultrasonic temperature readings are used for analysis.

With the averages calculated, the x, y, z components of wind velocity, and ultrasonic temperature were detrended and used to calculate the U, V, W, and ultrasonic temperature variances, the total turbulent kinetic energy of the air, and the covariances between U and V, U and W, V and W, U and temperature, V and temperature, and W and temperature. As discussed in Section 1.1.5 the systematic error can be calculated using (1.1) and the random error using (1.2) but first, the integral time scales must be determined. The integral time scales are calculated by finding the first zero-crossing of the auto-correlation or cross-auto-correlation functions. As discussed in Section 2.2.2 these functions can be calculated according to (2.22). As expected, most integral time scales are on the order of a few seconds, with the largest time scales being on the order of tens of seconds, seldom larger than 20 seconds. The variances and covariances are scaled using the systematic error for each point and measurement window.

5.3 Results

With the data collected and processed, analysis can be performed. The results are split into two parts, the averages over time and the turbulent statistics of each measured variable at each point in the room. All plots begin at 0800 and cover a full 24-hour interval.

5.3.1 Averages

The average wind components of the wind velocity vector in the room give a good picture of how the air is moving. Starting with the U component, as seen in Figure 5.2 there is minimal air movement at any of the measurement locations with the wind velocity vector component rarely exceeding an average of 0.1 $\mathrm{m\,s^{-1}}$. This is logical since the vents push air upwards near the windows resulting in a low x component of the wind velocity (U).

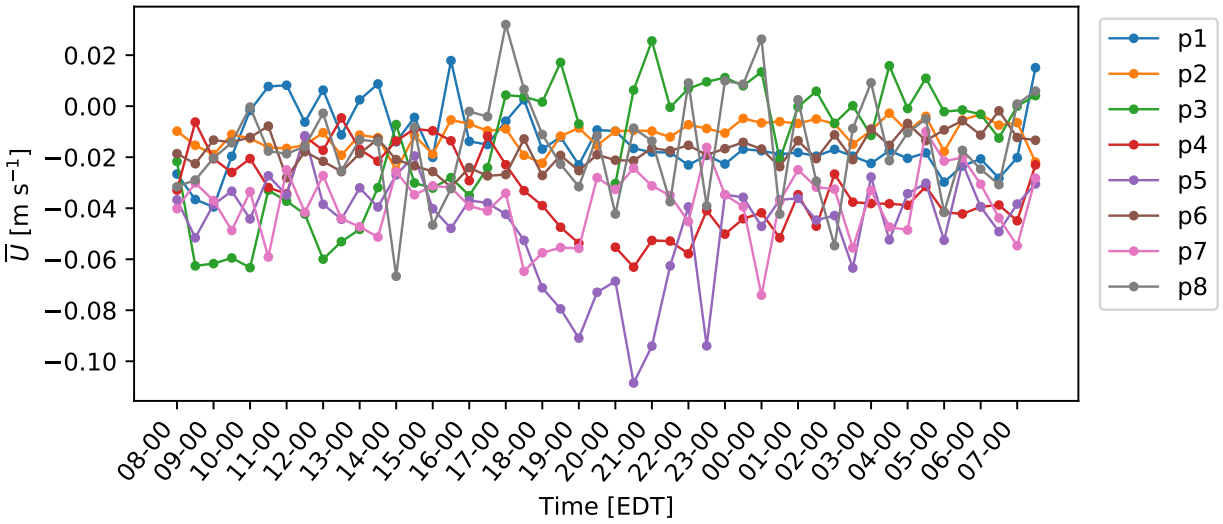

Figure 5.2: Average U component of the wind velocity vector over a diurnal cycle.

Figure 5.3 shows a large amount of air movement in the negative y direction near positions 7 and 8. In these locations, the increased wind velocity vector is due to the fan near the window blowing air from outside into the lab. During the day, the HVAC system is running and forcing air through the vents. Since the fan in the window is behind a vent, during the day between approximately 0800 and 1800, the air pushed by the fan interacts with the air being pushed up by the vent resulting in turbulence, and therefore, a lower horizontal velocity. In the evening and overnight, the wind speed is increased at position 8 as the HVAC turns off, and the fan can blow air directly towards ARES. Point 7 has a reduced wind velocity in the y direction after the HVAC

system turns off. This may be due to the turbulence created by the HVAC system widening the fan's area of effect. With the HVAC system turned off, the fan's area of effect was reduced, causing the airflow to miss point 7 while still reaching position 8. This effect is seen in the wind velocity component in the z direction (W) as well, although to a lesser degree, shown in Figure 5.4.

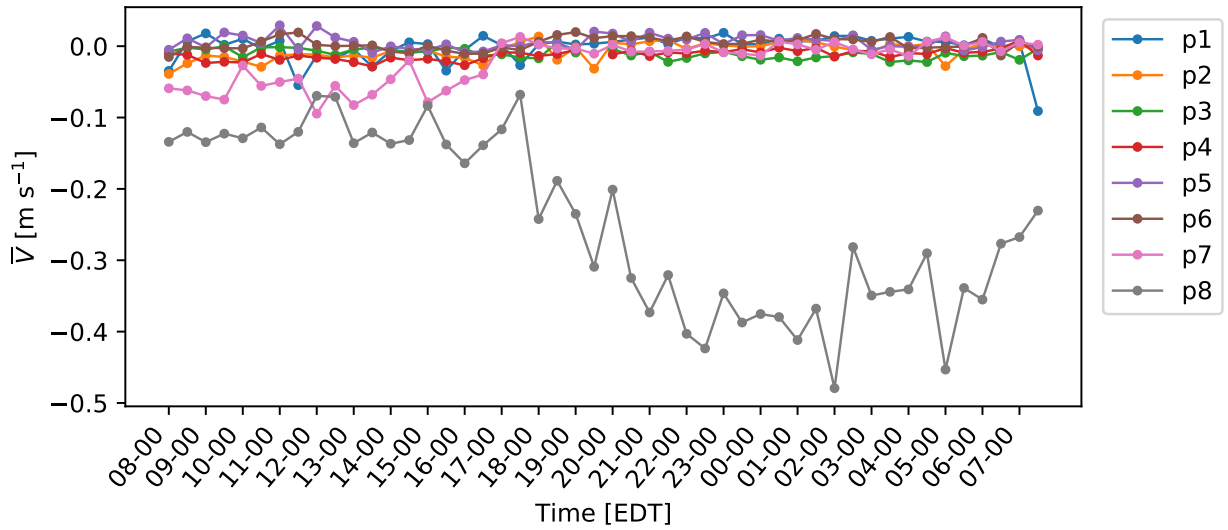

Figure 5.3: Average V component of the wind velocity vector over a diurnal cycle.

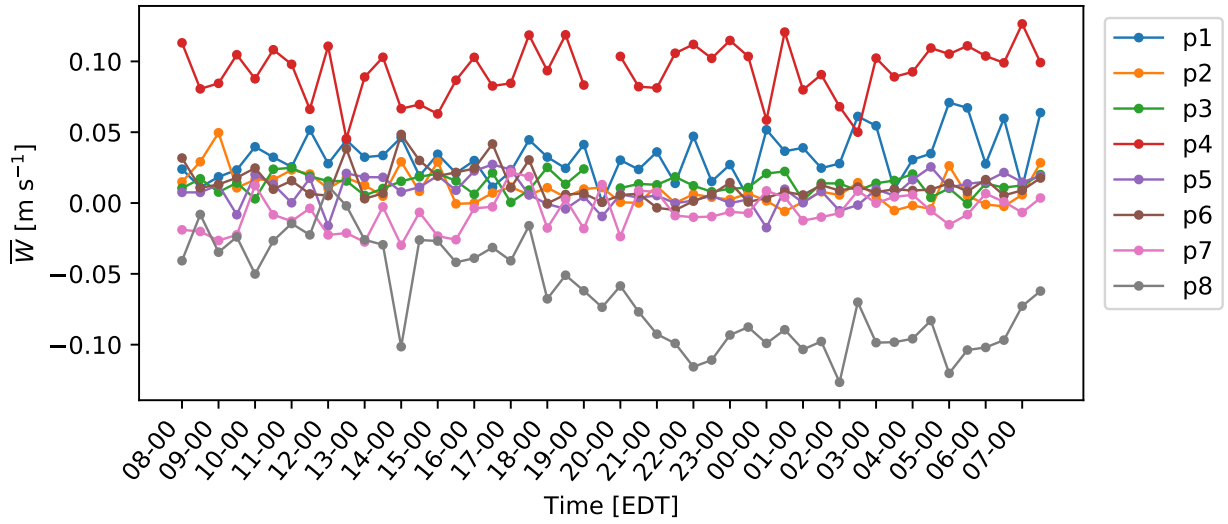

Figure 5.4: Average W component of the wind velocity vector over a diurnal cycle.

Similar to the x direction, all other measured locations have very low average V due to little air being input to the system through the HVAC system.

The air is quite stagnant in the z direction except for at positions 4 and 8. At position 4, there is an updraft likely caused by a heavily-used computer acting as a heat source. This is confirmed in Figure 5.5 as position 4 is warmer than the rest of the room. Position 8 experiences a downward draft, likely due to a combination of two factors. First, the fan in the window is tilted upwards, causing the air to reach the ceiling and then be redirected back down towards position 8. Additionally, cool air is being pulled in from outside, which tends to fall towards the floor, causing a larger downdraft at position 8. It should be noted that while there is a notable negative $\overline{W}$ at position 8, it is still less than a fifth of $\overline{V}$ in magnitude.

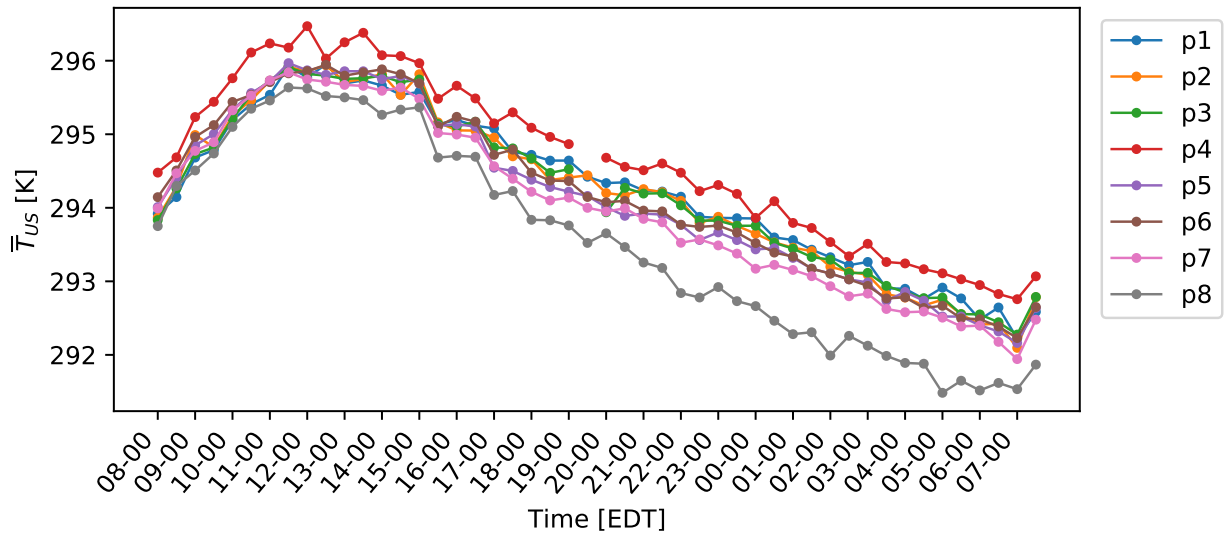

Figure 5.5: Average ultrasonic temperature over a diurnal cycle.

The total average wind speed $\overline{S}$ is calculated according to (2.17). As expected, position 8 has the largest total average wind speed by far, which is dominated by $\overline{V}$. Position 7 is close to position 8, and therefore also experiences some higher than average wind speeds. Position 4 has a slightly higher total average wind speed due to the rising air from the heat source. In general, wind speeds

in the room are measured around or below 0.1 $\mathrm{m\,s^{-1}}$ throughout the day in the room with the exception of position 8.

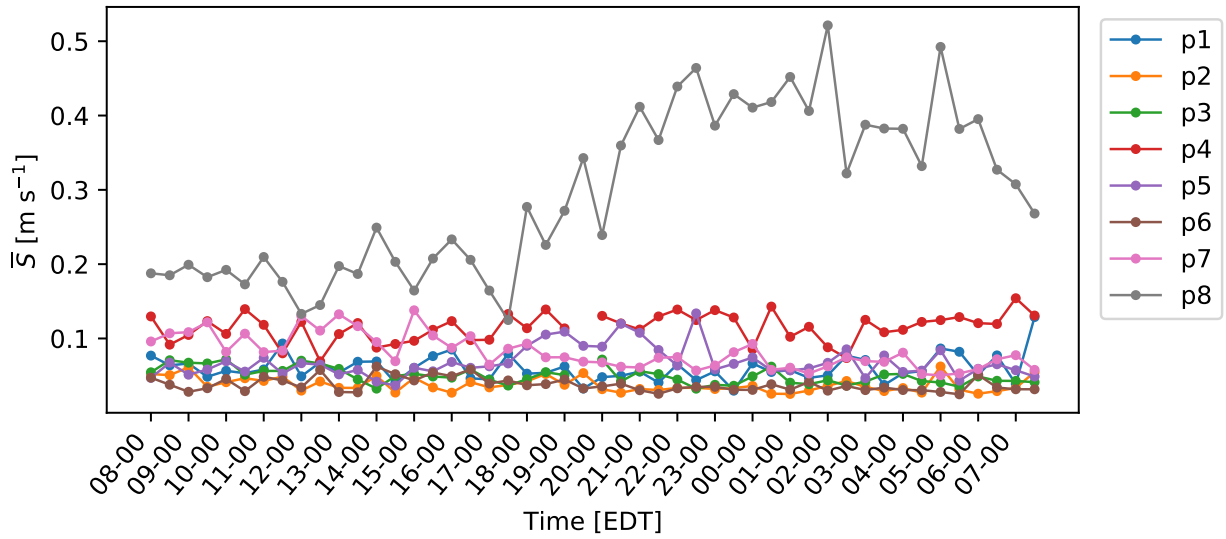

Figure 5.6: Average wind speed over a diurnal cycle.

Two measurements were made for temperature, one using the ultrasonic anemometer temperature reading and one from a platinum resistance thermometer in the HMP60 sensor. As previously mentioned, the ultrasonic anemometer measures temperature without any thermal mass, allowing for measurements of each location's instantaneous temperature, giving it an advantage over the slower measurement time of the HMP60 sensor and is used for further analysis. A plot of the ultrasonic temperature and temperature read by the HMP60 sensor can be found in Figures 5.5 and 5.7 respectively. It should be noted that the ultrasonic temperature was calibrated using data from the HMP60 platinum resistance thermometer.

The temperature at position 4 is a fraction of a Kelvin higher than the rest of the room, likely due to one or more nearby computers generating heat during use. As expected, position 8 has a lower average temperature than the rest of the room due to cool air being blown into the room from the outside with a fan. The temperature time series exhibits a strong diurnal cycle with the room

warming from 0730 until approximately 1200 as the room is warmed during the daytime and the air being pulled into the room from outside rises in temperature. It should be noted that during these experiments, the heating system was turned off, so any warming could not be caused by the HVAC system directly. Around approximately 1800, the temperature begins to drop and continues to do so through the night as the fan blows cool air into the room from outside, slowly lowering the average temperature. There is a large drop in temperature at the 1530 mark; however, this is due to the two different days tests were run on and not a physical phenomenon. This sudden change at 1530 can be seen in other graphs as well.

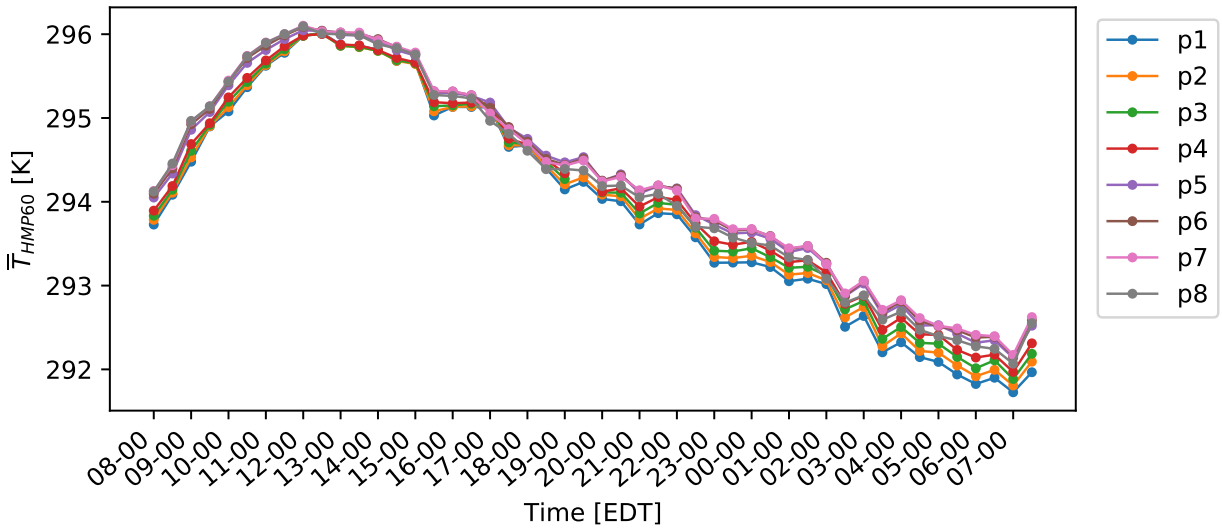

Figure 5.7: Average HMP60 temperature over a diurnal cycle.

Finally, the average relative humidity is plotted in Figure 5.8. The relative humidity at all points is fairly consistent throughout the diurnal cycle, only varying by a few percent throughout the day. There is a drop in the relative humidity during the morning hours, likely due to the rising temperature, which, using psychrometric considerations, will decrease the relative humidity. As the day moves past 1200, however, the relative humidity moves back to near 33% as the room balances with the rest of the building through the HVAC system. There is a slight divergence of

relative humidities during the night, with the positions closer to the open window having a slightly lower relative humidity than positions further from the window. This is caused by a difference in relative humidity inside and outside. The outside air is both cooler and has a lower relative humidity which manifests as a gradient in relative humidity across the room. The highest average relative humidity is found at position 1, which may be caused by the experimenter introducing extra moisture into the air.

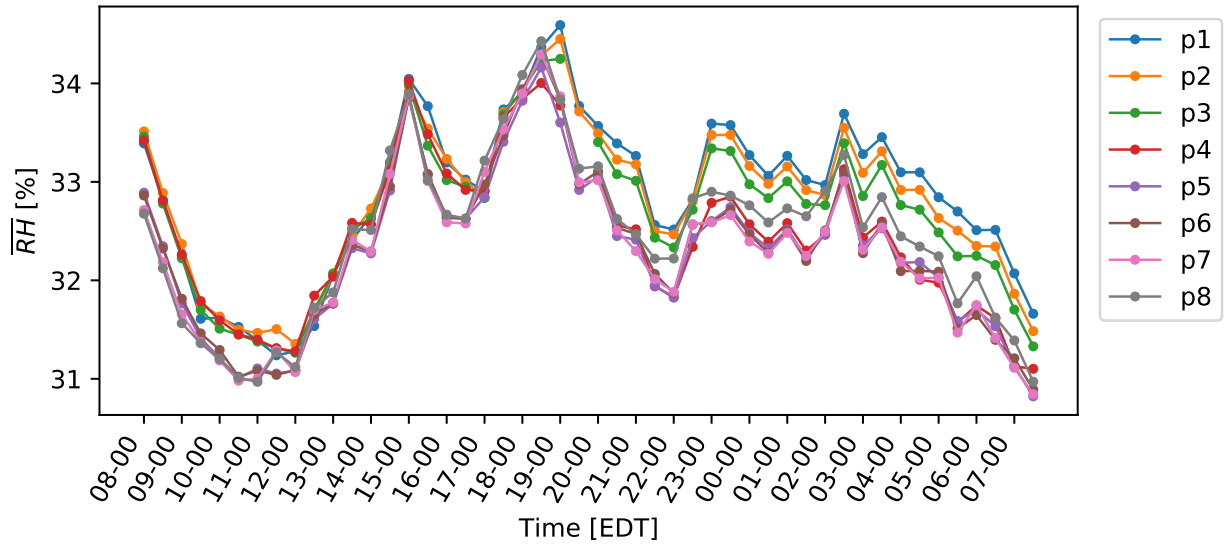

Figure 5.8: Average relative humidity over a diurnal cycle.

5.3.2 Variances and Covariances

A few interesting trends show up when looking at each variable's variances (or covariance of two variables). It should be noted that as discussed in section 2.2.1 the variance in wind velocity vector components is normalized using the average total wind speed $\overline{S}$ over the entire room during the time interval.

Variance in U, shown in Figure 5.9, confirms that the air in the room is fairly slow-moving except for the area where the fan is placed. Turbulence from the fan creates an area of higher

variance around positions 7 and 8. This effect is seen again in the variance of V, although in this case, it is much more strongly seen since this is the direction in which the fan is blowing air. Interestingly position 7 has minimal variance in V likely since it is just barely in the area of effect from the fan during the day and not in the area of effect during the evening and overnight. This manifests as position 7 having a normalized variance in V of approximately 0.4 during the day while dropping nearly to 0 at night.

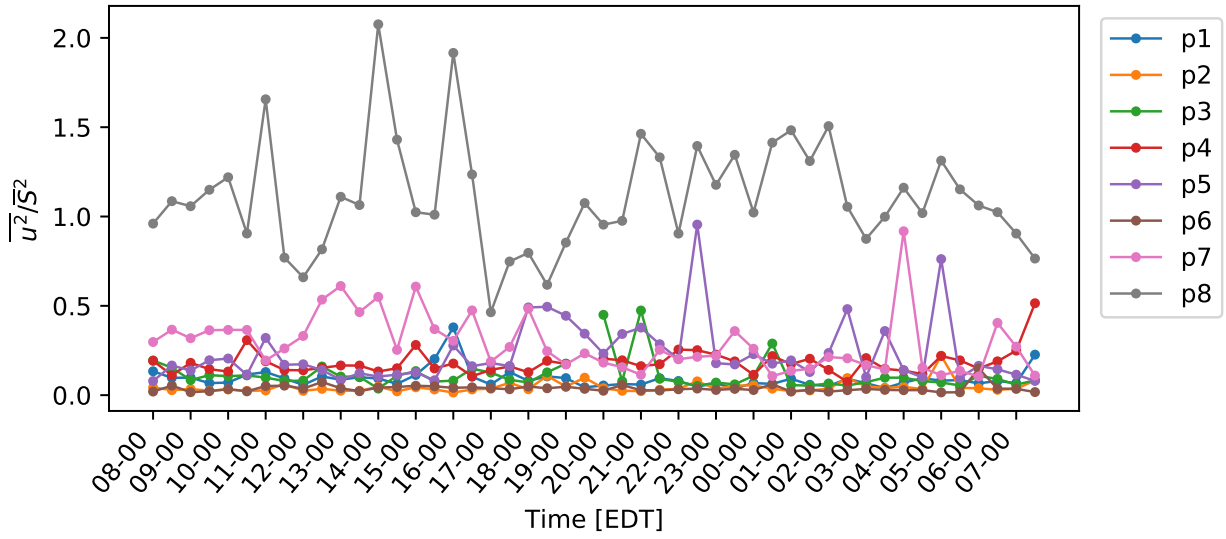

Figure 5.9: Normalized wind velocity vector variance in the x direction over a diurnal cycle.

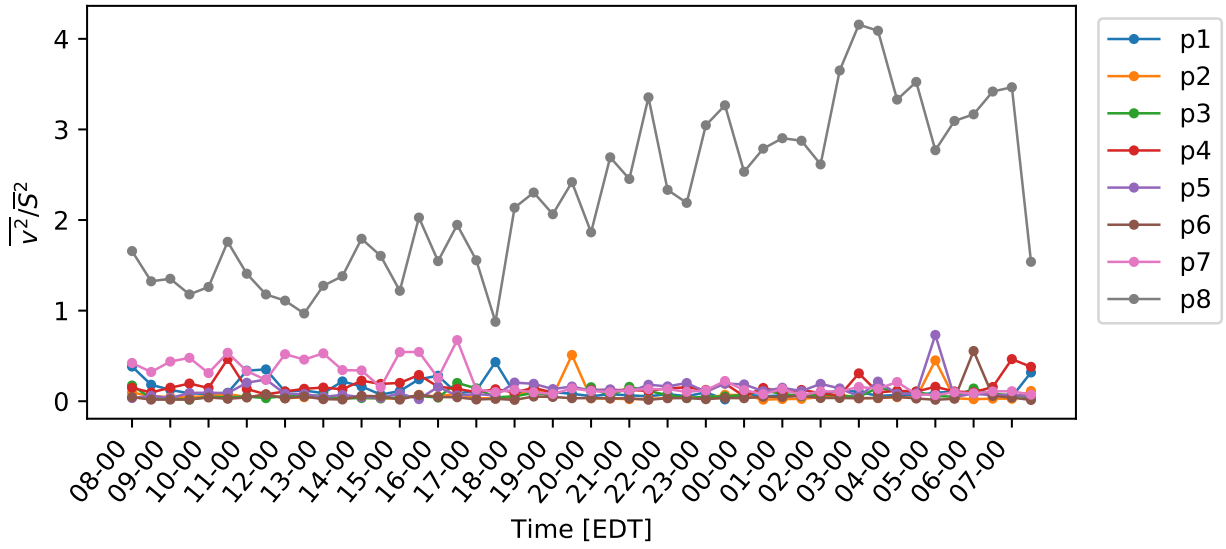

Figure 5.10: Normalized wind velocity variance in the y direction over a diurnal cycle.

Variance in W is significantly smaller than in U and V. Position 8 sees significant variance due to the fan pushing air unevenly through the location. Position 7 sees some variance in W due to the fan but to a lesser degree. Position 4 shows greater variance due to rising hot air from the computer at the workstation. Position 1 has seen slightly higher than usual variances in all directions, likely due to the experimenter sitting at a computer approximately two feet from the platform causing an increase in variance due to their movement.

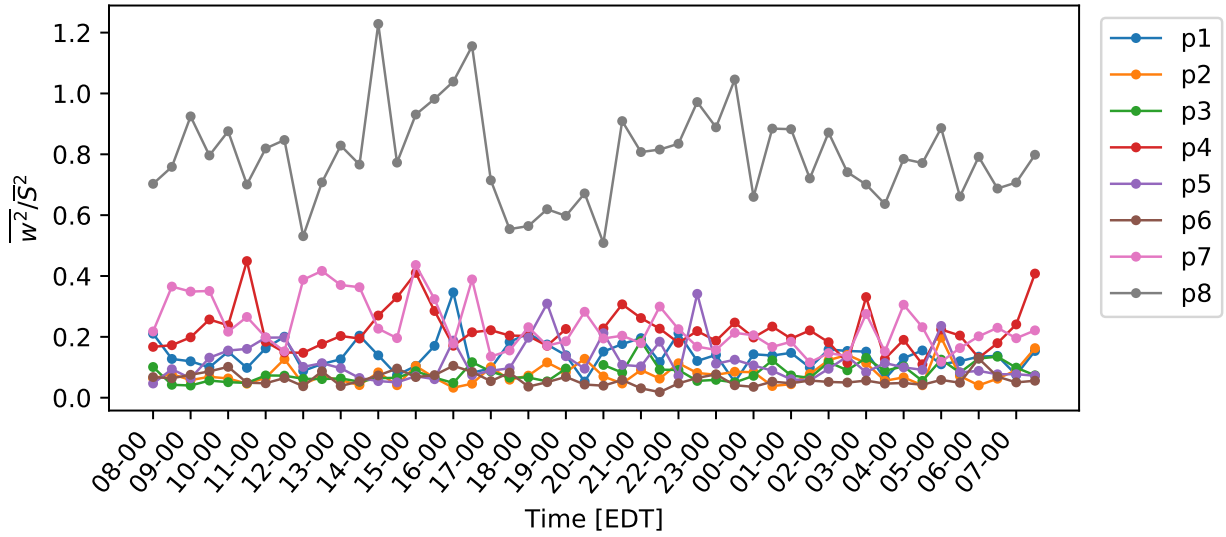

Figure 5.11: Normalized wind velocity vector variance in the z direction over a diurnal cycle.

The variance of temperature, shown in Figure 5.12, remains fairly constant throughout the workday due to the HVAC system keeping the room's air well-mixed; however, in the evening and overnight, the cold air being forced into the room mixes with the warm air creating a larger variance in temperature. This effect is seen strongly at position 8, where the mixing first occurs. There is also a notable temperature variance at position 4 because the heated air rising to the sensor mixes with the cooler air creating warm and cool parcels of air that move past the sensor.

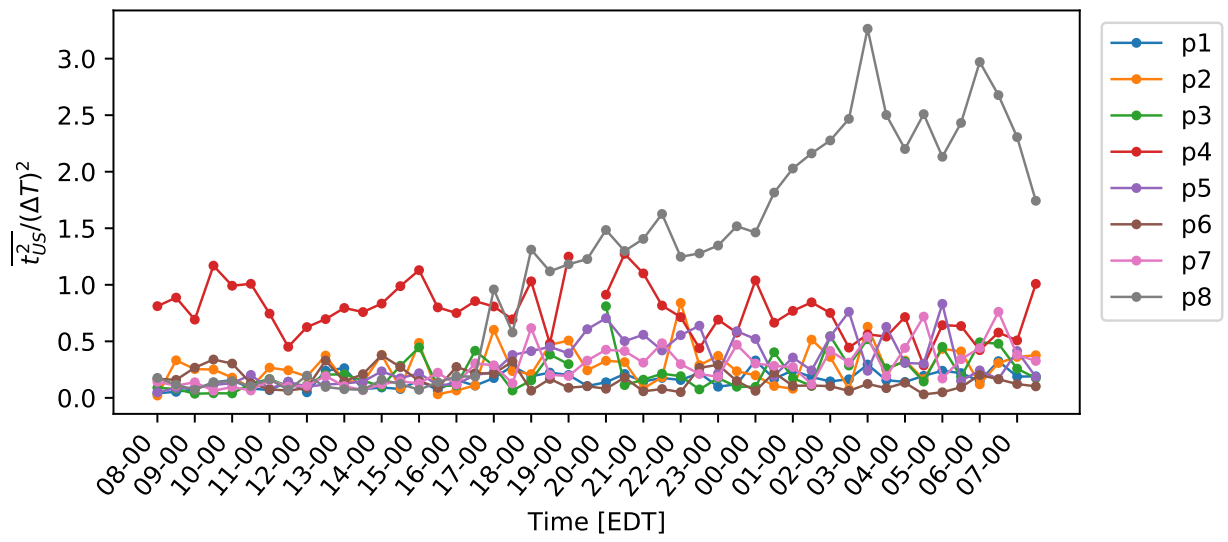

Figure 5.12: Normalized ultrasonic temperature variance over a diurnal cycle.

The turbulent kinetic energy (k) provides a good representation of how turbulent the air is. As shown in Figure 5.13, most of the room is fairly stable with a normalized k below 0.5; however, positions 7 and 8 show large normalized k values due to the fan inducing turbulence in the room. Interestingly it appears that the HVAC system reduces the k about position 8 slightly as during the day, the average normalized k ranges between 1.5 and 2, while at night, the normalized k is above 2. There is also a slight increase in normalized k at position 4 due to the hot air rising and creating small eddy currents.

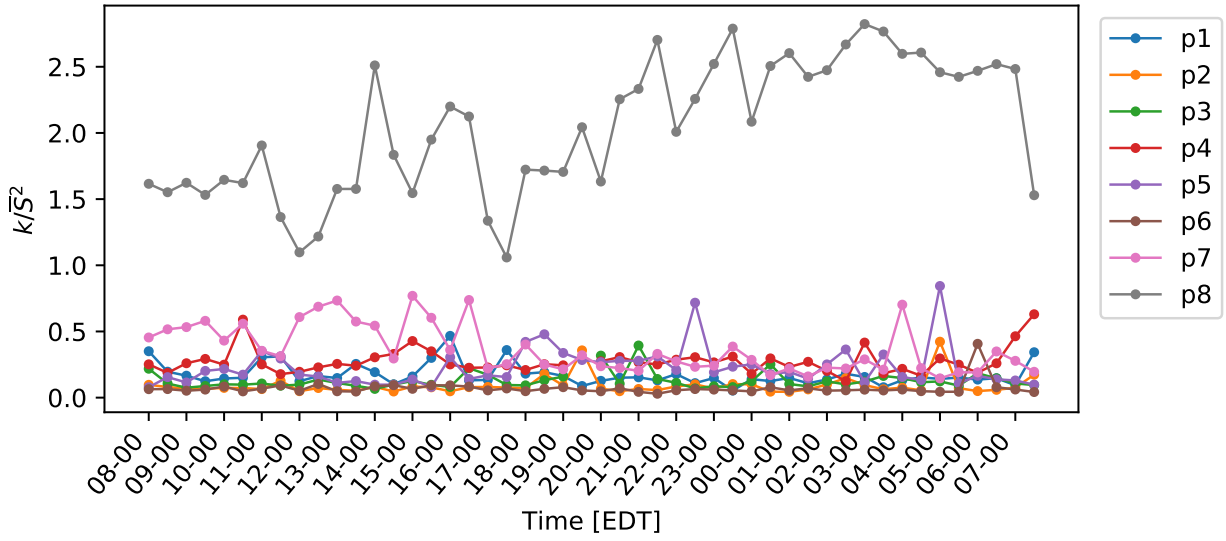

Figure 5.13: Normalized turbulent kinetic energy over a diurnal cycle.

Most of the covariances continue to tell the same story, that at position 8 there is notable covariance between the vertical component of the wind velocity vector and temperature, and that position 7 sees some of the same covariance, but to a lesser degree. For this reason, a number of the covariance graphs are included in Appendix C.1. The covariance between wind velocity in the z direction and temperature strongly shows the existence of a heat source at position 4. Due to the rising air and increased temperature, the covariance between W and T is raised.

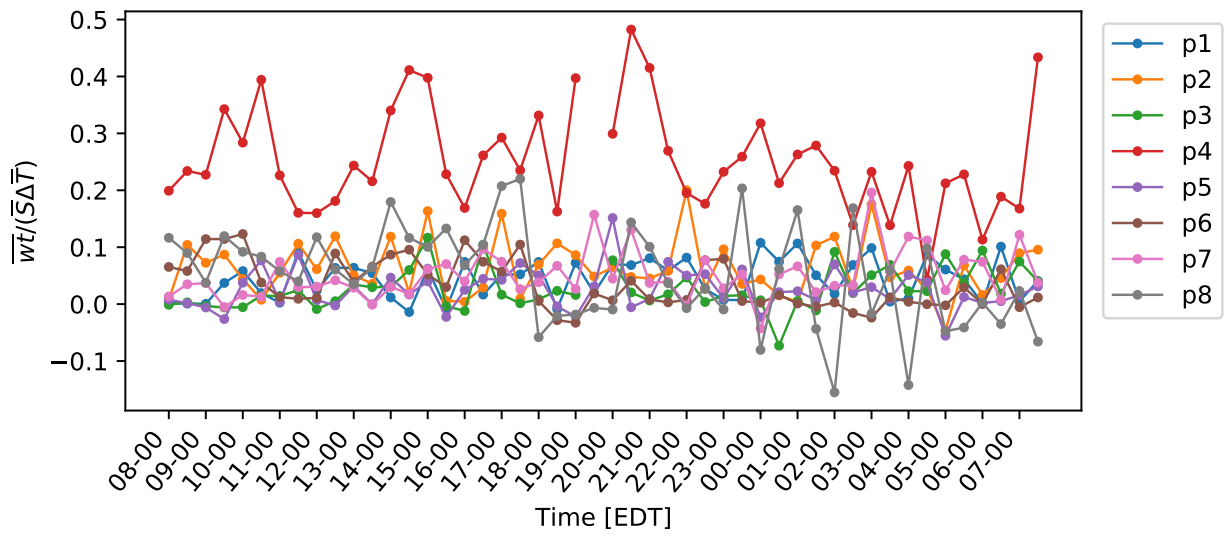

Figure 5.14: Normalized covariance between the wind velocity in the z direction and ultrasonic temperature over a diurnal cycle.

5.4 Thermal Comfort

The thermal comfort of people working in the lab is of particular interest, and with the data collected, the predicted mean vote (PMV) and predicted percent dissatisfied (PPD) are calculated at each point over the course of a diurnal cycle. In typical office spaces, the area is only used during the day; however, since this lab is occupied by graduate students who often run on unconventional schedules, data from 0600 to 2359 should be investigated. In some cases, people are in the lab throughout the night, so to cover those cases, the PMVs and PPDs are plotted for the full 24 hour period. It should be noted that the mean radiant temperature is assumed to be the same as the ambient air temperature for all cases. While not perfectly accurate, this assumption is reasonable as discussed in Chapters 1 and 2.

A number of different metabolic rates and clothing levels are investigated. At positions 1, 4, 6, and 7, people are typically sitting down working on a computer corresponding to a metabolic

rate of 1.1 met. At positions 2, 3, 5, and 8, people are typically standing working on a whiteboard or walking at a leisurely pace for a short period of time, resulting in a metabolic rate of 1.7 met. These tests were conducted as the weather was beginning to reach cold temperatures, resulting in different clothing levels depending on the person. Three clothing levels are investigated: a person wearing trousers with a short-sleeve shirt, socks and shoes giving a clothing level of 0.57 clo; a person wearing sweat pants and a long sleeve sweater with socks and shoes giving a clothing level of 0.74 clo; and a person wearing trousers, a long sleeve shirt, and a jacket with socks and shoes giving a clothing level of 0.94 clo. These different cases are listed in Table 5.1.

Table 5.1: List of metabolism (met) and clothing (clo) levels used for each test case.

Test Case	1	2	3	4	5	6
Metabolism	1.1	1.1	1.1	1.7	1.7	1.7
Clothing	0.57	0.74	0.96	0.57	0.74	0.96

Beginning with case 1, where the subjects are sitting at their desks working on a computer wearing light clothing, during the day the PMV is approximately between -0.6 and -1.4, which corresponds to the room being slightly cool. As is supported by the PPD in Figure 5.16, during normal working hours, most people would find the temperature in the room acceptable; however, into the evening and through the night the room becomes quite cold, leading to the majority of people finding the room cold between 0000 and 0730. The room's coolness was noted by the experimenter causing him to wear a sweater from 1900 to 0900.

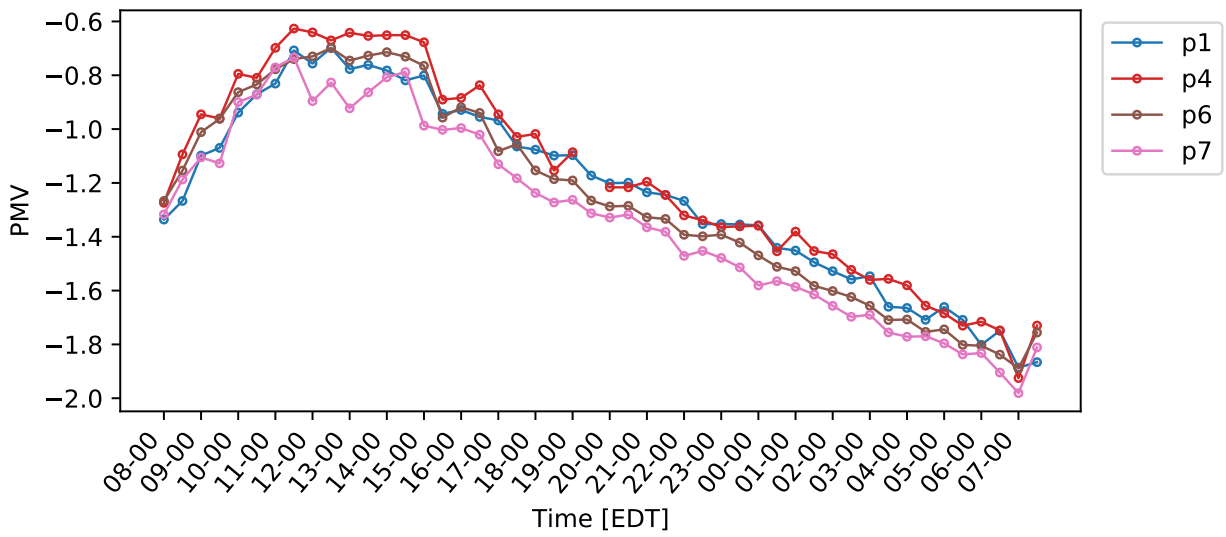

Figure 5.15: PMV with metabolism = 1.1 met and clothing = 0.57 clo over a diurnal cycle.

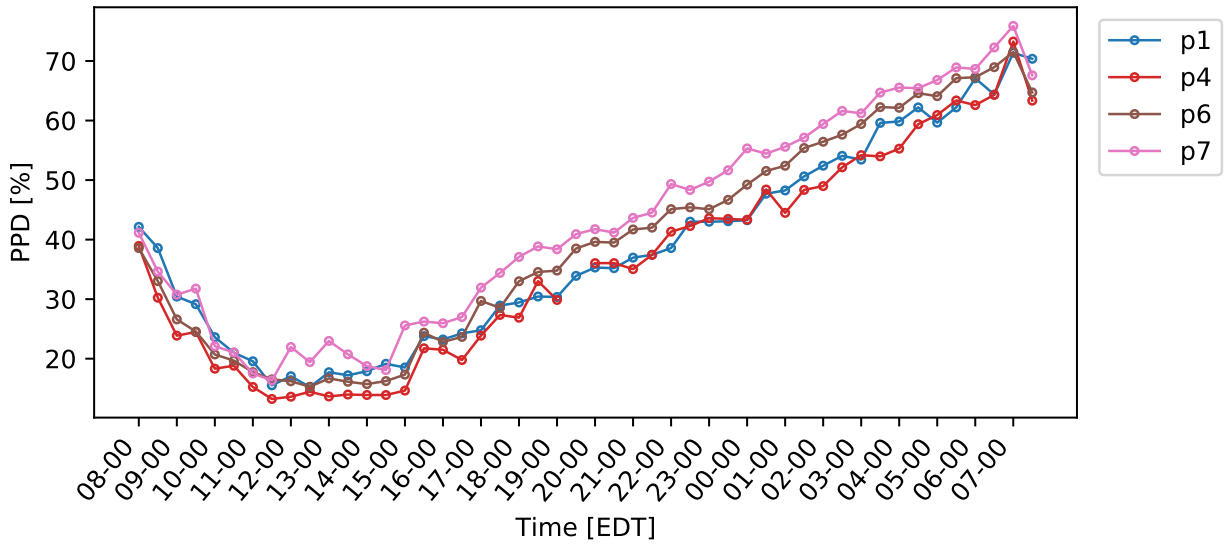

Figure 5.16: PDD with metabolism = 1.1 met and clothing = 0.57 clo over a diurnal cycle.

Increasing clothing to a long sleeve sweater increases the PMV during working hours by approximately -0.3, as seen in Figure 5.17. In turn, the PPD during working hours is decreased to approximately 10% nearing an optimal environment. The shape of the PMV strongly resembles the

shape of the mean temperature in Figure 5.5 indicating that the main factor to thermal comfort in low wind speed environments is the ambient air temperature. While the experimenter was wearing a sweater in the evening and through the night in order to remain comfortable, at approximately 0300 the experimenter became too cold for comfort.

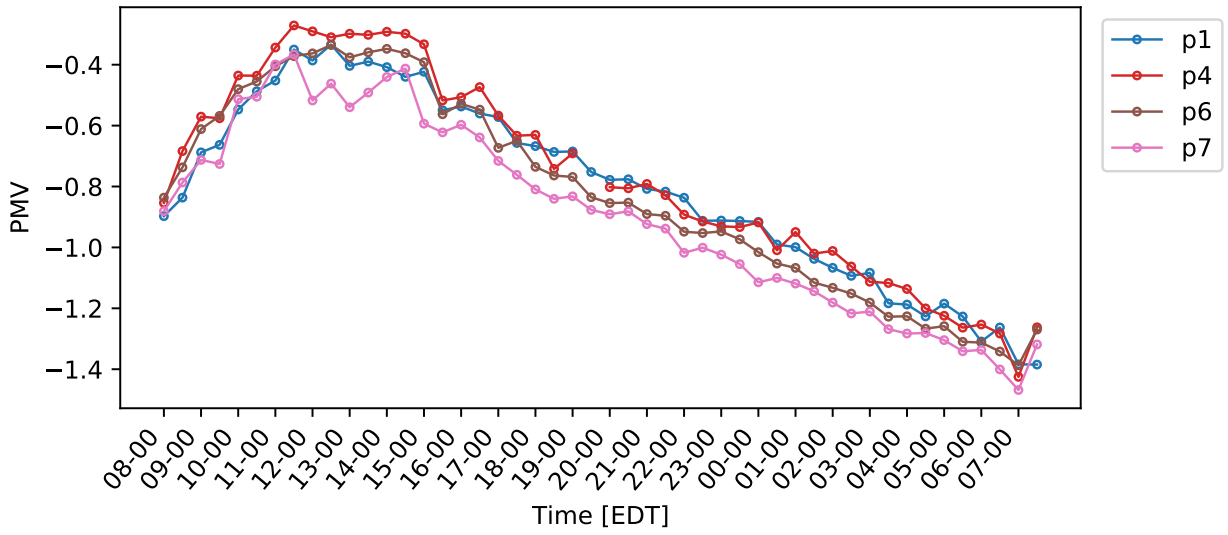

Figure 5.17: PMV with metabolism = 1.1 met and clothing = 0.74 clo over a diurnal cycle.

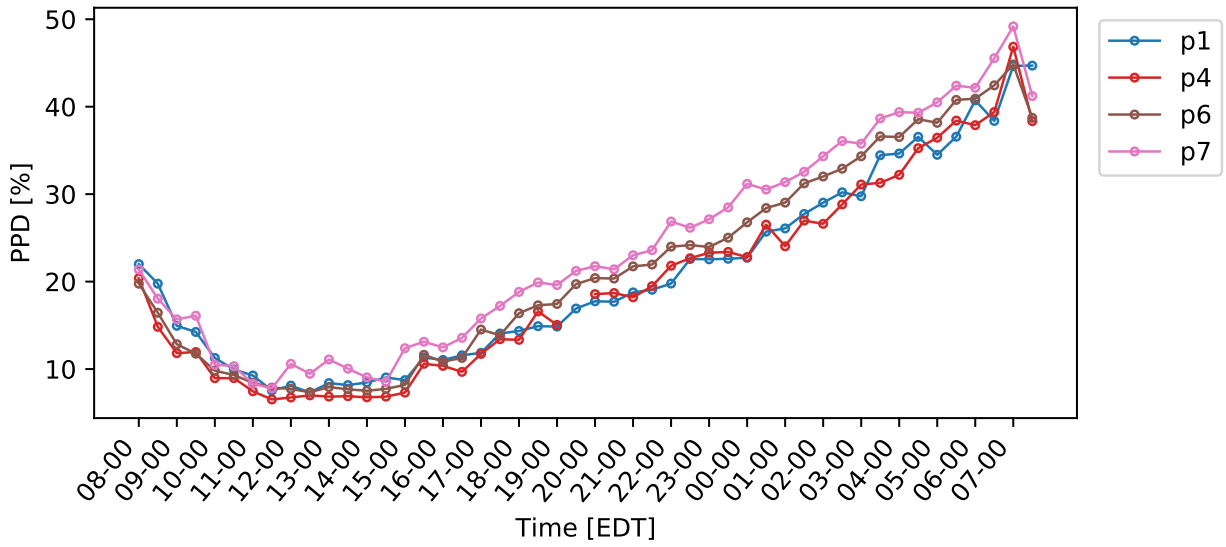

Figure 5.18: PDD with metabolism = 1.1 met and clothing = 0.74 clo over a diurnal cycle.

Increasing the clothing again to wearing a jacket results in the best thermal comfort, with the PMV reaching neutrality in the afternoon. The PPD is reduced to sub 10% throughout the day and remains there until approximately 2000, well after most people would have left the lab. Even better, the PPD reaches 5% between 1000 and 0300, indicating an optimal thermal comfort environment. Unsurprisingly, overnight the PPD increases significantly but still remains comfortable to the majority of people. Comparing different clothing levels shows a wide range of thermal comforts are possible simply by adjusting what is worn. This puts an emphasis on keeping environmental variables stable and allowing individuals to change their comfort through clothing choice instead of through cooling and heating of the environment.

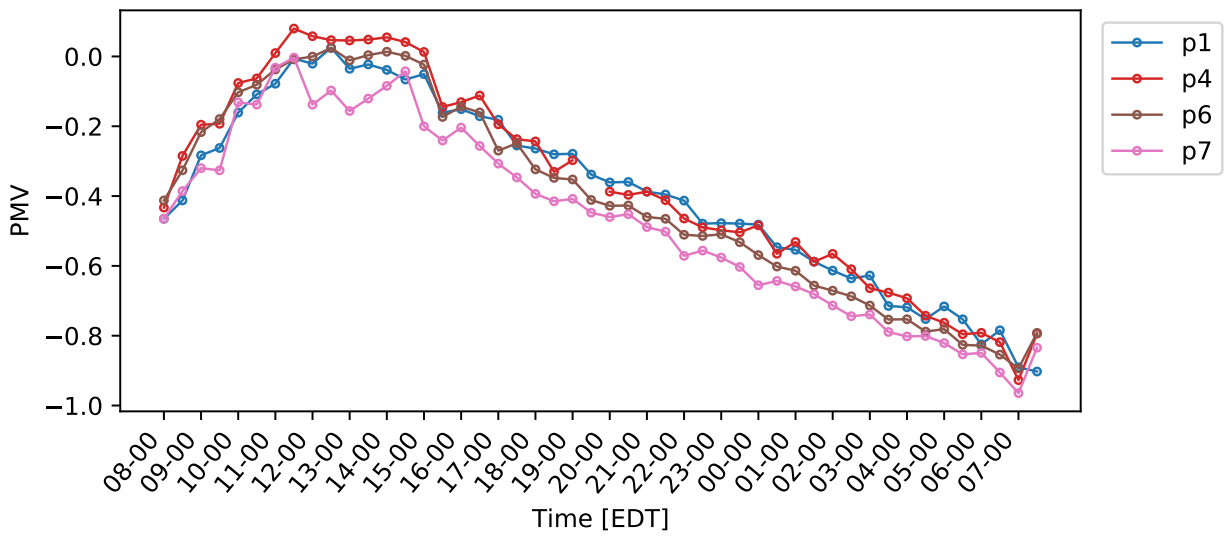

Figure 5.19: PMV with metabolism = 1.1 met and clothing = 0.96 clo over a diurnal cycle.

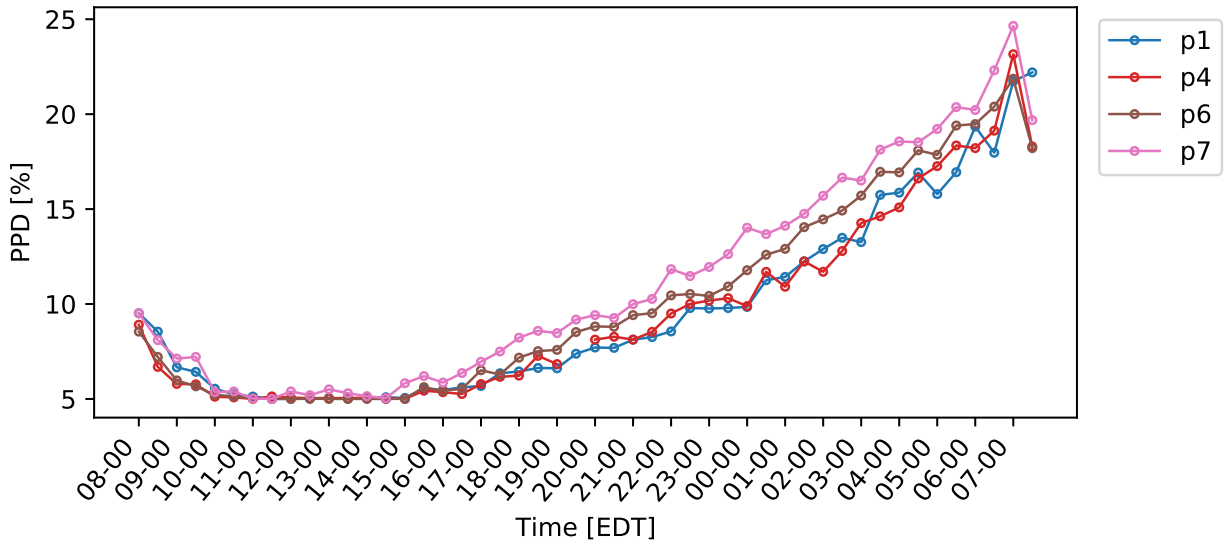

Figure 5.20: PDD with metabolism = 1.1 met and clothing = 0.96 clo over a diurnal cycle.

Cases 4-6 show that the PMV significantly increases due to an increase in metabolism. Figures 5.21 through 5.26 show the PMV and PPD with a metabolism of 1.7 met. The PMV at positions 2, 3, and 5 are slightly warm during the day but slowly reduce overnight as the room cools. Position

8 has a significantly lower thermal comfort due to the fan creating a high wind speed during the day and producing higher wind speeds and a low-temperature zone during the evening and night. On average, position 8 has a PMV 0.25 to 0.5 lower than other areas of the room where people are typically moving about. During the day, the PPD at position 8 is near or at 5%, the lowest the PPD can be, indicating excellent thermal comfort; however, in the evening the cool air significantly reduces the PPD. Positions 2, 3, and 5 are fairly comfortable while moving throughout the day and night, never rising above 10%. Interestingly, the PPD is higher at these positions than point 8 during the day, indicating that a reduction in temperature in the room may result in better thermal comfort across the room as a whole. This is confirmed by the existence of the fan in the window, which was placed there due to lab members finding the room too hot during the day.

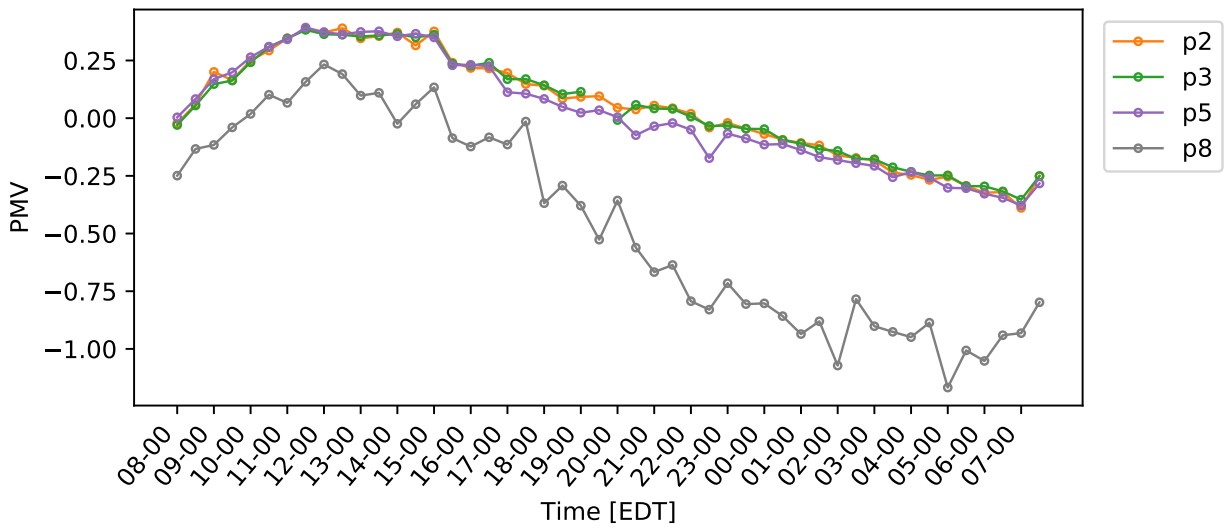

Figure 5.21: PMV with metabolism = 1.7 met and clothing = 0.57 clo over a diurnal cycle.

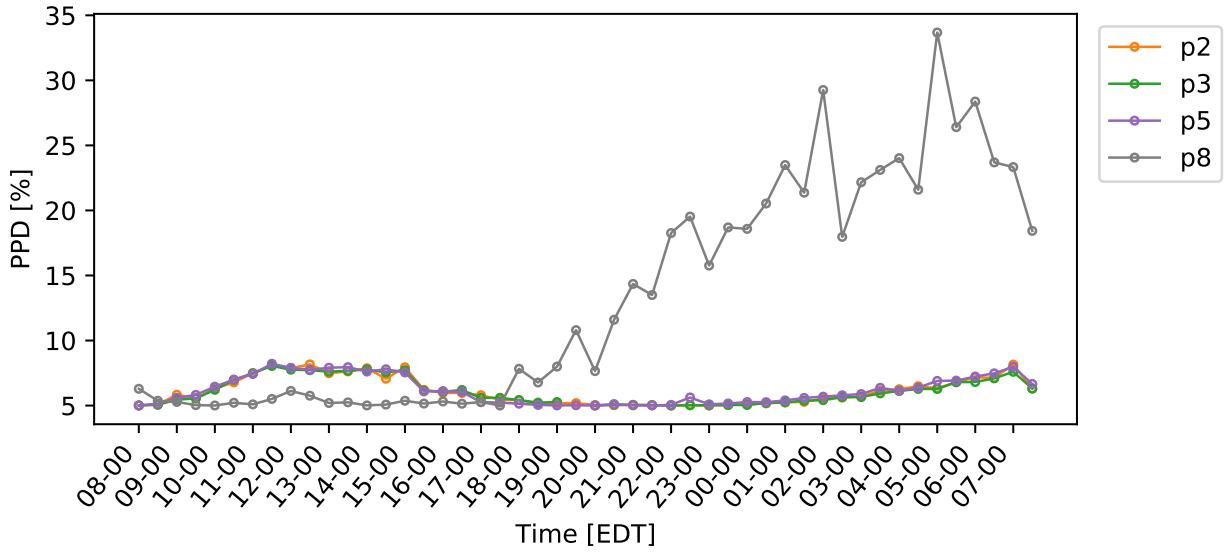

Figure 5.22: PDD with metabolism = 1.7 met and clothing = 0.57 clo over a diurnal cycle.

Increasing the level of clothing also increases the PMV across the board and, in fact, worsens the PPD at all points, with the PPD reaching 12% and 18% in cases 5 and 6 during the day, respectively. These results can be seen in Figures 5.24 and 5.26. The experimenter wore clothing with a clothing value of around 0.74 during the night and kept himself moving about in order to remain comfortable. This behaviour is confirmed in Figure 5.24 where the PPD tends toward 5% overnight while not directly in the fan's path.

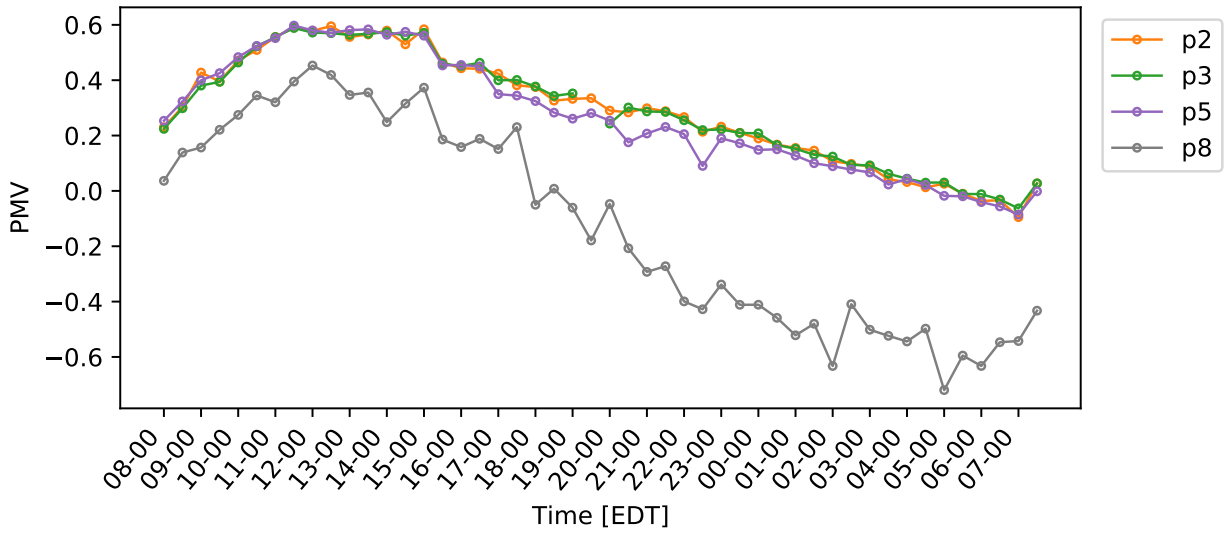

Figure 5.23: PMV with metabolism = 1.7 met and clothing = 0.74 clo over a diurnal cycle.

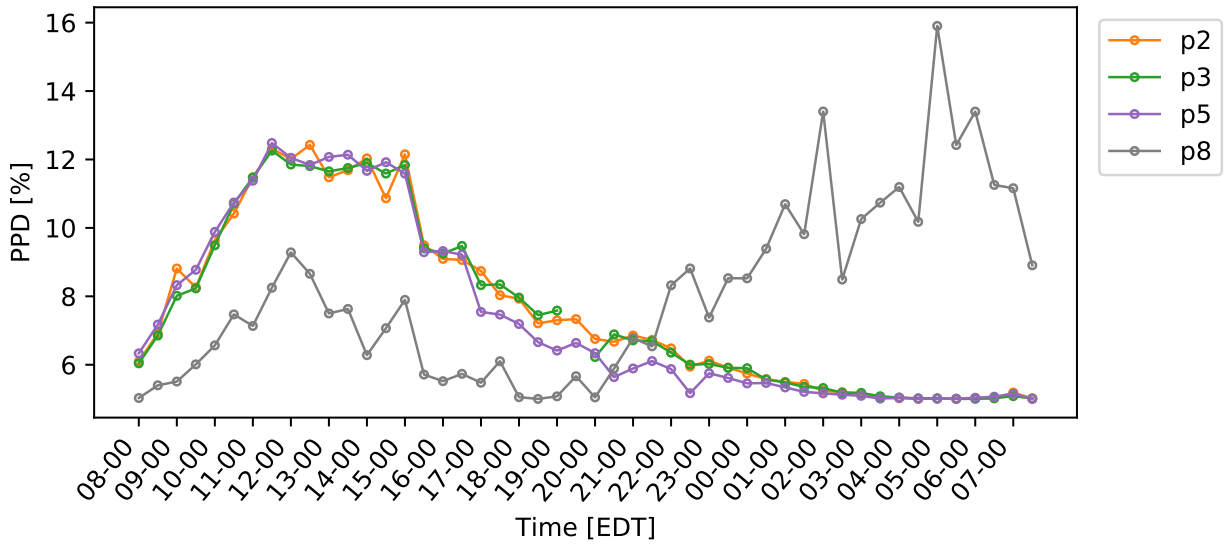

Figure 5.24: PDD with metabolism = 1.7 met and clothing = 0.74 clo over a diurnal cycle.

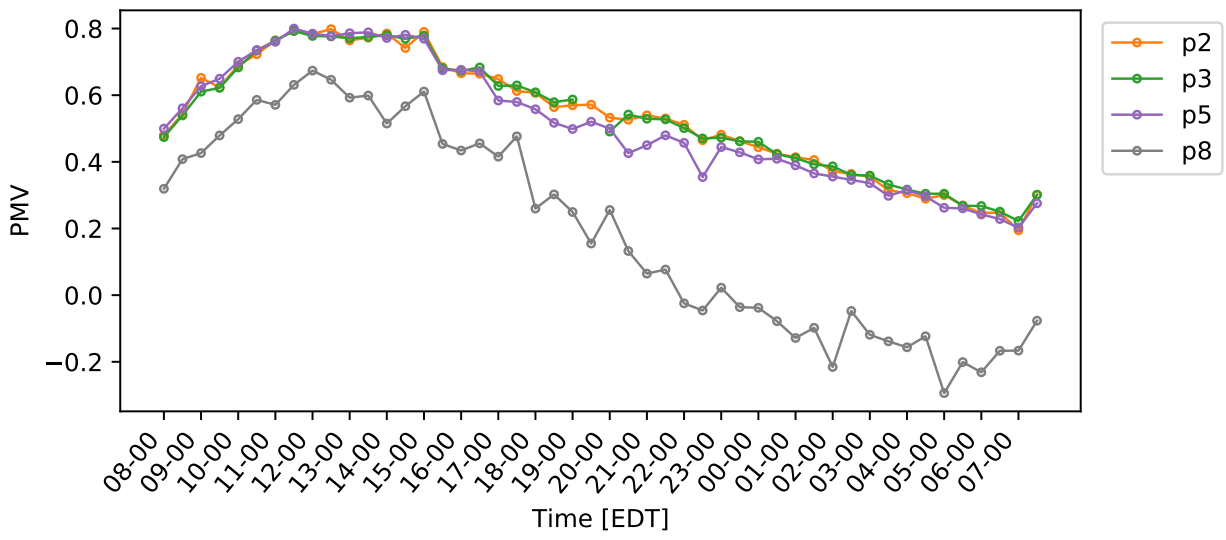

Figure 5.25: PMV with metabolism = 1.7 met and clothing = 0.96 clo over a diurnal cycle.

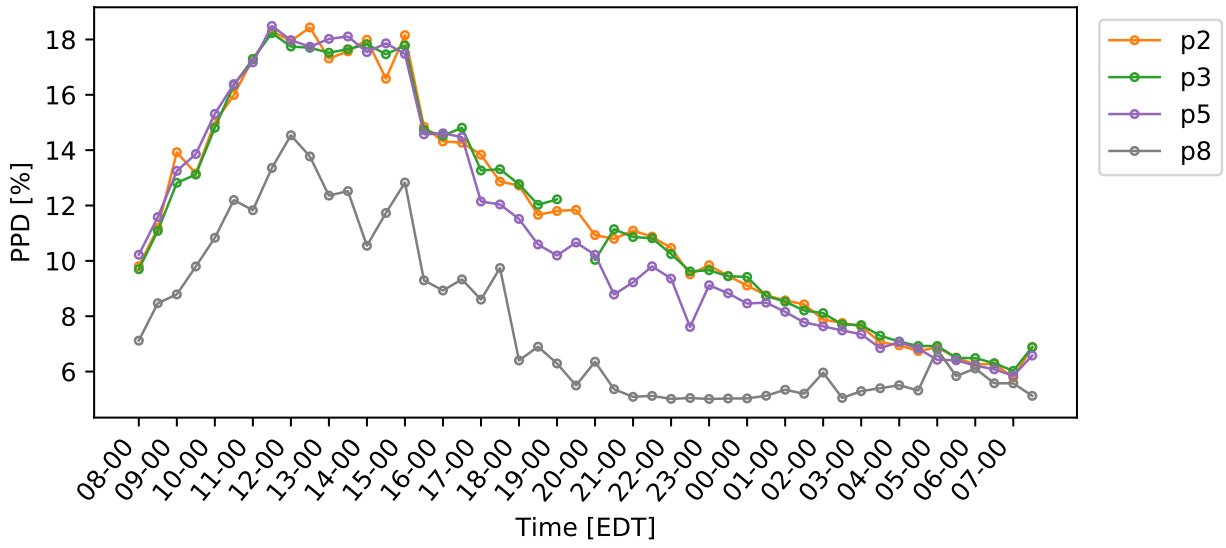

Figure 5.26: PDD with metabolism = 1.7 met and clothing = 0.96 clo over a diurnal cycle.

5.5 Platform viability

These tests show that while improvements can be made on the sensing platform, it effectively quantifies the indoor environment. After collecting and processing data, small adjustments can be made to the environmental control system of a room to make sure thermal comfort is achieved. In the Mechatronics lab, high temperatures during the day have caused the need for a cooling fan to be placed in the room; however, the single point of cooling causes uneven thermal comfort. A better solution would be to increase cooling to the room from the HVAC system, lowering the temperature evenly and creating a more comfortable environment.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Sensing of the indoor environment has traditionally been accomplished using a large number of static sensors causing measurements of the indoor environment to be prohibitively expensive. To reduce cost, the Autonomous Robotic Environmental Sensor (ARES) was developed, providing a low-cost, easy-to-use, robust alternative to static sensors. ARES was developed from the ground up, starting by prototyping a three-wheel drive omniwheel robot with custom hardware and subsequently building the platform to carry any arrangement of environmental sensors.

In order to accurately position ARES, a feedback linearization controller and a sliding mode controller were developed based on the derived robot kinematics with wheel slip. Through experimentation with a VICON system, the sliding mode controller was found to provide significantly better performance than the feedback linearization controller, with a position RMSE of approximately 4 mm in the x and y directions. This RMSE is 4-5 times lower than the feedback linearization controller. In addition, the sliding mode controller reaches the desired state significantly faster than the feedback linearization controller, allowing ARES to spend more time taking measurements of the environment and less time moving between measurement points.

ARES was used to measure environmental variables over a full diurnal cycle at eight positions in the mechatronics lab at the University of Guelph. Outfitted with an ultrasonic anemometer, temperature sensor, and relative humidity sensor, ARES measured environmental variables used to predict thermal comfort. The robot acted autonomously when taking measurements, only requiring small positional adjustments between measurement periods and changing of batteries. Statistical analysis of the collected environmental data helped identify areas with higher than average wind speeds, low temperatures, and the location of heat sources. Data collected was also used to predict thermal comfort and identified areas of the room that a group of people would find too cold or too hot throughout the day. This was shown most clearly by the significant decrease in thermal comfort near the back of the room due to a window fan creating a cold draft and the gradient in thermal comfort caused by the cooling of one side of the room.

The creation of a robust, modular, autonomous environmental sensing platform has been accomplished through the development of ARES. The use of omniwheels allows ARES to navigate tight spaces and avoid obstacles easily. Due to the platform's modularity, it can be used for any number of sensing tasks, from thermal comfort to gas source localization to indoor environmental quality with minimal hardware adjustments. It was shown that ARES is capable of taking meaningful measurements of the environment, a task that would otherwise require multiple sets of expensive sensors.

6.2 Future Work

ARES is a flexible platform that can be used in many indoor environmental sensing scenarios, enabling the platform for multiple uses in future research. If used for further research in thermal comfort, a mean radiant temperature sensor should be added to eliminate the small error caused by assuming the mean radiant temperature is the same as the ambient air temperature. By including more anemometers at different heights, comfort factors such as floor drafts can also be measured

and quantified.

The robot can perform tasks over a longer period of time without human intervention if more sensors such as gyroscopes or Light Detection and Ranging (LIDAR) sensors are used to more accurately determine the robot's position. Additionally, the use of higher-speed motors would allow for larger rooms to be measured in the same period of time. Finally, more complex controllers such as non-linear model predictive control could be implemented, creating a more robust control system resulting in more precise movement.

References

M. Ahmadi-Baloutaki and A. A. Aliabadi. A Very Large-Eddy Simulation Model Using a Reductionist Inlet Turbulence Generator and Wall Modelling for Stable Atmospheric Boundary Layers. *Fluid Dynamics*, 56(3):413–432, 2021. doi: 10.1134/S0015462821020026.

AL-Taharwa. A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment. *Journal of computer science*, 4(4):341–344, 2008. doi: 10.3844/jcssp.2008.341.344.

V. Alakshendra and S. S. Chiddarwar. A Robust Adaptive Control of Mecanum Wheel Mobile Robot: Simulation and Experimental Validation. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5606–5611, Daejeon, Korea, November 2016. doi: 10.1109/IROS.2016.7759824.

A. A. Aliabadi, S. N. Rogak, K. H. Bartlett, and S. I. Green. Preventing Airborne Disease Transmission: Review of Methods for Ventilation Design in Health Care Facilities. *Advances in Preventive Medicine*, 2011:1–21, 2011. doi: 10.4061/2011/124064.

A. A. Aliabadi, R. M. Staebler, M. Liu, and A. Herber. Characterization and Parametrization of Reynolds Stress and Turbulent Heat Flux in the Stably-Stratified Lower Arctic Troposphere Using Aircraft Measurements. *Boundary-Layer Meteorology*, 161(1):99–126, 2016. doi: 10.1007/s10546-016-0164-7.

ASHRAE. Standard 55-2004 (2004) Thermal environmental conditions for human occupancy. *Atlanta: American Society of Heating Refrigerating and Air-Conditioning Engineers Inc*, 2017.

M. J. Barrett and D. K. Hollingsworth. On the calculation of length scales for turbulent heat transfer correlation. *Journal of Heat Transfer*, 123(5):878–883, 2001. doi: 10.1115/1.1391277.

C. Benton, F. Bauman, and U. Fountain, M. A Field Measurement System for the Study of Thermal Comfort. *ASHRAE Transaction*, 96(1):623–633, 1990.

A. V. Borisov, A. A. Kilin, and I. S. Mamaev. Dynamics and control of an omniwheel vehicle. *Regular and Chaotic Dynamics*, 20(2):153–172, 2015. doi: 10.1134/S1560354715020045.

A. Bramanta, A. Virgono, and R. E. Saputra. Control system implementation and analysis for omniwheel vehicle. In *International Conference on Control, Electronics, Renewable Energy, and Communications, Proceedings*, pages 265–270, Yogyakarta, Indonesia, January 2017. doi: 10.1109/ICCEREC.2017.8226711.

A. Bulińska, Z. Popiołek, and Z. Buliński. Experimentally validated CFD analysis on sampling region determination of average indoor carbon dioxide concentration in occupied space. *Building and Environment*, 72:319–331, 2014. doi: 10.1016/j.buildenv.2013.11.001.

J. Canny and J. Reif. New Lower Bound Techniques for Robot Motion Planning Problems. In *Annual Symposium on Foundations of Computer Science (Proceedings)*, pages 49–60, Los Angeles, USA, July 1987. IEEE. doi: 10.1109/sfcs.1987.42.

Y. Chen, H. Cai, Z. Chen, and Q. Feng. Using multi-robot active olfaction method to locate time-varying contaminant source in indoor environment. *Building and Environment*, 118:101–112, 2017. doi: 10.1016/j.buildenv.2017.03.030.

M. Dawe, P. Raftery, J. Woolley, S. Schiavon, and F. Bauman. Comparison of mean radiant and air temperatures in mechanically-conditioned commercial buildings from over 200,000 field and laboratory measurements. *Energy and Buildings*, 206:109582, 2020. doi: 10.1016/j.enbuild.2019.109582.

M. De Villiers and N. S. Tlale. Development of a control model for a four wheel mecanum vehicle. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 134(1):4–9, 2012. doi: 10.1115/1.4005273.

G. Dozier, A. Esterline, A. Homaifar, and M. Bikdash. Hybrid evolutionary motion planning via visibility-based repair. In *IEEE International Conference on Evolutionary Computation*, pages 507–511, Indianapolis, USA, August 1997. doi: 10.1109/ICEC.1997.592363.

P. O. Fanger. Thermal comfort: Analysis and applications in environmental engineering. *Applied Ergonomics*, 3(3):181, sep 1972. doi: 10.1016/S0003-6870(72)80074-7.

V. Földváry Ličina, T. Cheung, H. Zhang, R. de Dear, T. Parkinson, E. Arens, C. Chun, S. Schiavon, M. Luo, G. Brager, P. Li, S. Kaam, M. A. Adebamowo, M. M. Andamon, F. Babich, C. Bouden, H. Bukovianska, C. Candido, B. Cao, S. Carlucci, D. K. Cheong, J. H. Choi, M. Cook, P. Cropper, M. Deuble, S. Heidari, M. Indraganti, Q. Jin, H. Kim, J. Kim, K. Konis, M. K. Singh, A. Kwok, R. Lamberts, D. Loveday, J. Langevin, S. Manu, C. Moosmann, F. Nicol, R. Ooka, N. A. Oseland, L. Pagliano, D. Petráš, R. Rawal, R. Romero, H. B. Rijal, C. Sekhar, M. Schweiker, F. Tartarini, S. ichi Tanabe, K. W. Tham, D. Teli, J. Toftum, L. Toledo, K. Tsuzuki, R. De Vecchi, A. Wagner, Z. Wang, H. Wallbaum, L. Webb, L. Yang, Y. Zhu, Y. Zhai, Y. Zhang, and X. Zhou. Development of the ASHRAE Global Thermal Comfort Database II. *Building and Environment*, 142:502–512, 2018. doi: 10.1016/j.buildenv.2018.06.022.

Y. Fukazawa and H. Ishida. Estimating gas-source location in outdoor environment using mobile robot equipped with gas sensors and anemometer. In *Proceedings of IEEE Sensors*, pages 1721–1724, Christchurch, New Zealand, 2009. IEEE. doi: 10.1109/ICSENS.2009.5398495.

J. Haberl, H. Davies, B. Owens, and B. Hunn. ASHRAE's New Performance Measurement Protocols for Commercial Buildings. Technical report, Energy Systems Laboratory, Berlin, Germany, October 2008.

D. Heinzerling, S. Schiavon, T. Webster, and E. Arens. Indoor environmental quality assessment models: A literature review and a proposed weighting and classification scheme. *Building and Environment*, 70:210–222, 2013. doi: 10.1016/j.buildenv.2013.08.027.

J. H. Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, USA, 1 edition, 1992.

M. Jin, S. Liu, S. Schiavon, and C. Spanos. Automated mobile sensing: Towards high-granularity agile indoor environmental quality monitoring. *Building and Environment*, 127:268–276, 2018. doi: 10.1016/j.buildenv.2017.11.003.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, August 1995.

A. Kilin, P. Bozek, Y. Karavaev, A. Klekovkin, and V. Shestakov. Experimental investigations of a highly maneuverable mobile omniwheel robot. *International Journal of Advanced Robotic Systems*, 14(6):1–9, 2017. doi: 10.1177/1729881417744570.

S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, Edmonton, Canada, July 2002.

D. H. Lenschow, J. Mann, and L. Kristensen. How long is long enough when measuring fluxes and other turbulence statistics? *Journal of Atmospheric & Oceanic Technology*, 11(3):661–673, 1994. doi: 10.1175/1520-0426(1994)011⟨0661:HLILEW⟩2.0.CO;2.

M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research*, 28(8):933–945, 2009. doi: 10.1177/0278364909340445.

L.-C. Lin and H.-Y. Shih. Modeling and Adaptive Control of an Omni-Mecanum-Wheeled Robot. *Intelligent Control and Automation*, 04(02):166–179, 2013. doi: 10.4236/ica.2013.42021.

W. Liu, Y. Zhang, and Q. Deng. The effects of urban microclimate on outdoor thermal sensation and neutral temperature in hot-summer and cold-winter climate. *Energy and Buildings*, 128: 190–197, 2016. doi: 10.1016/j.enbuild.2016.06.086.

Y. Liu, R. L. Williams, and J. J. Zhu. Integrated control and navigation for omni-directional mobile robot based on trajectory linearization. In *Proceedings of the American Control Conference*, pages 2153–2158, New York City, USA, 2007. doi: 10.1109/ACC.2007.4282967.

T. Lozano-Pérez. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983. doi: 10.1109/TC.1983.1676196.

T. Lozano-Pérez and M. A. Wesley. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of the ACM*, 22(10):560–570, 1979. doi: 10.1145/359156.359164.

J. Nicole and K. McCartney. Smart controls and thermal comfort project. SCATs final report. Technical report, Oxford, Headington, UK, 2000.

P. V. Nielsen. Fifty years of CFD for room air distribution. *Building and Environment*, 91:78–90, 2015. doi: 10.1016/j.buildenv.2015.02.035.

J. D. Posner, C. R. Buchanan, and D. Dunn-Rankin. Measurement and prediction of indoor air flow in a model room. *Energy and Buildings*, 35(5):515–526, 2003. doi: 10.1016/S0378-7788(02)00163-9.

R. D. Puits, C. Resagk, and A. Thess. Thermal boundary layers in turbulent Rayleigh-Bénard convection at aspect ratios between 1 and 9. *New Journal of Physics*, 15:013040, 2013. doi: 10.1088/1367-2630/15/1/013040.

Y. Q. Qin, D. B. Sun, N. Li., and Y. G. Cen. Path planning for mobile robot using the particle swarm optimization with mutation operator. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, number August, pages 2473–2478, Shanghai, China, January 2004. doi: 10.1109/icmlc.2004.1382219.

P. Raja and S. Pugazhenthi. Optimal path planning of mobile robots: A review. *International Journal of the Physical Sciences*, 7(9):1314–1320, 2012. doi: 10.5897/ijps11.1745.

M. Reggente, A. Mondini, G. Ferri, B. Mazzolai, A. Manzi, M. Gabelletti, P. Dario, and A. J. Lilienthal. The DustBot system: Using mobile robots to monitor pollution in pedestrian area. *Chemical Engineering Transactions*, 23(June 2014):273–278, 2010. doi: 10.3303/CET1023046.

S. Schiavon, B. Yang, Y. Donner, V. W.-C. Chang, and W. W. Nazaroff. Thermal comfort, perceived air quality, and cognitive performance when personally controlled air movement is used by tropically acclimatized persons. *Indoor Air*, 27(3):690–702, 2017. doi: 10.1111/ina.12352.

G. Schiller, E. Arens, F. Bauman, C. Benton, M. Fountain, and T. Doherty. A field study of thermal environments and comfort in office building. *ASHRAE Transactions*, 94 Part 2, 1988.

X. Shan and W.-Z. Lu. An integrated approach to evaluate thermal comfort in air-conditioned large-space office. *Science and Technology for the Built Environment*, 0(0):1–15, 2020. doi: 10.1080/23744731.2020.1796420.

R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, and R. C. Arkin. *Introduction to Autonomous Mobile Robots*. MIT Press, Cambridge, Massachusetts, 2nd edition, 2011.

M. P. Spilak, T. Sigsgaard, H. Takai, and G. Zhang. A comparison between temperature-controlled laminar airflow device and a room air-cleaner in reducing exposure to particles while asleep. *PLoS ONE*, 11:1–21, 2016. doi: 10.1371/journal.pone.0166882.

D. Stonier, S. H. Cho, S. L. Choi, N. S. Kuppuswamy, and J. H. Kim. Nonlinear slip dynamics for an omniwheel mobile robot platform. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2367–2372, Rome, Italy, April 2007. doi: 10.1109/ROBOT.2007.363673.

Z. Sun, H. Xie, J. Zheng, Z. Man, and D. He. Path-following control of Mecanum-wheels omnidirectional mobile robots using nonsingular terminal sliding mode. *Mechanical Systems and Signal Processing*, 147:107128, 2020. doi: 10.1016/j.ymssp.2020.107128.

H. Takimoto, A. Sato, J. F. Barlow, R. Moriwaki, A. Inagaki, S. Onomura, and M. Kanda. Particle Image Velocimetry Measurements of Turbulent Flow Within Ouutdoor and Indoor Urban Scale Models and Flushing Motions in Urban Canopy Layers. *Boundary-Layer Meteorol*, 140:295–314, 2011. doi: http://dx.doi.org.subzero.lib.uoguelph.ca/10.1007/s10546-011-9612-6.

K. J. Udupa and I. Murthy. New Concepts for Three-Dimensional Shape Analysis. *IEEE Transactions on Computers*, C-26(10):1043–1049, 1977.

H. Widyantara, M. Rivai, and D. Purwanto. Gas Source Localization Using an Olfactory Mobile Robot Equipped With Wind Direction Sensor. In *2018 International Conference on Computer Engineering, Network and Intelligent Multimedia, CENIM 2018 - Proceeding*, pages 66–70, Surabaya, Indonesia, May 2018. doi: 10.1109/CENIM.2018.8711381.

D. E. Williams. Low Cost Sensor Networks: How Do We Know the Data Are Reliable? *ACS Sensors*, 4(10):2558–2565, 2019. doi: 10.1021/acssensors.9b01455.

L. Zhang, J. Kim, and J. Sun. Energy modeling and experimental validation of four-wheel mecanum mobile robots for energy-optimal motion control. *Symmetry*, 11:1372–1386, 2019. doi: 10.3390/sym11111372.

W. Zhang, K. Hiyama, S. Kato, and Y. Ishida. Building energy simulation considering spatial temperature distribution for nonuniform indoor environment. *Building and Environment*, 63:89–96, 2013. doi: 10.1016/j.buildenv.2013.02.007.

B. Zhao, X. Li, and Q. Yan. A simplified system for indoor airflow simulation. *Building and Environment*, 38(4):543–552, 2003. doi: 10.1016/S0360-1323(02)00182-8.

D. Zhu and J.-C. Latombe. New Heuristic Algorithms for Efficient Hierarchical Path Planning. *IEEE Transaction on Robotics and Automation*, 7(1):9–20, 1991. doi: 10.1109/70.68066.

A. A. Zobova and Y. V. Tatarinov. Free and controlled motions of an omniwheel vehicle. *Moscow University Mechanics Bulletin*, 63(6):146–150, 2008. doi: 10.3103/S0027133008060034.

Appendix A

Chapter 3 Supplement

This appendix contains schematics for all custom PCBs developed for ARES. All files associated with the PCBs can be found at https://github.com/dyerbm/OmniWheel.

A.1 Electrical Schematics

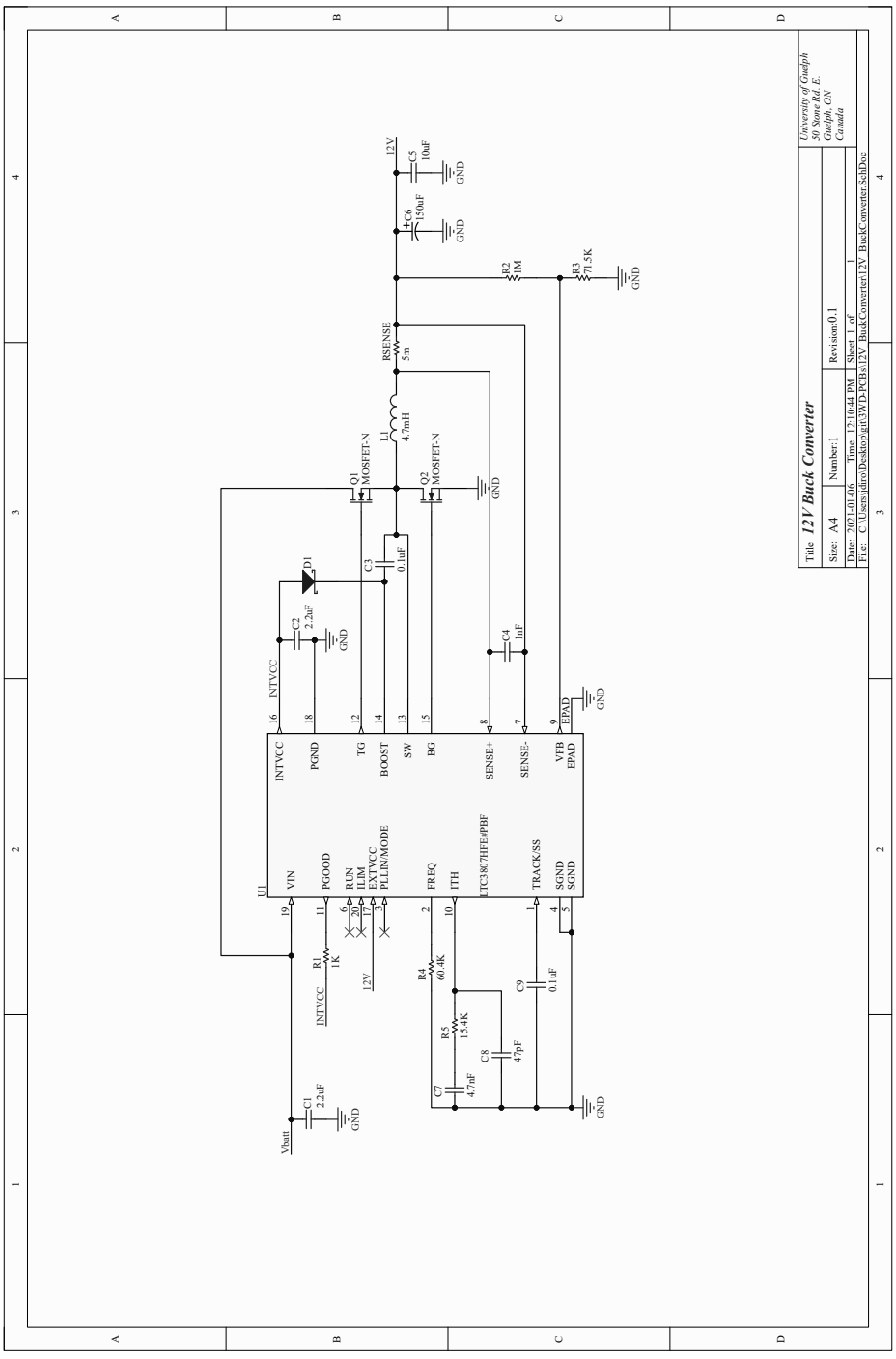

Figure A.1: 12V buck converter schematic.

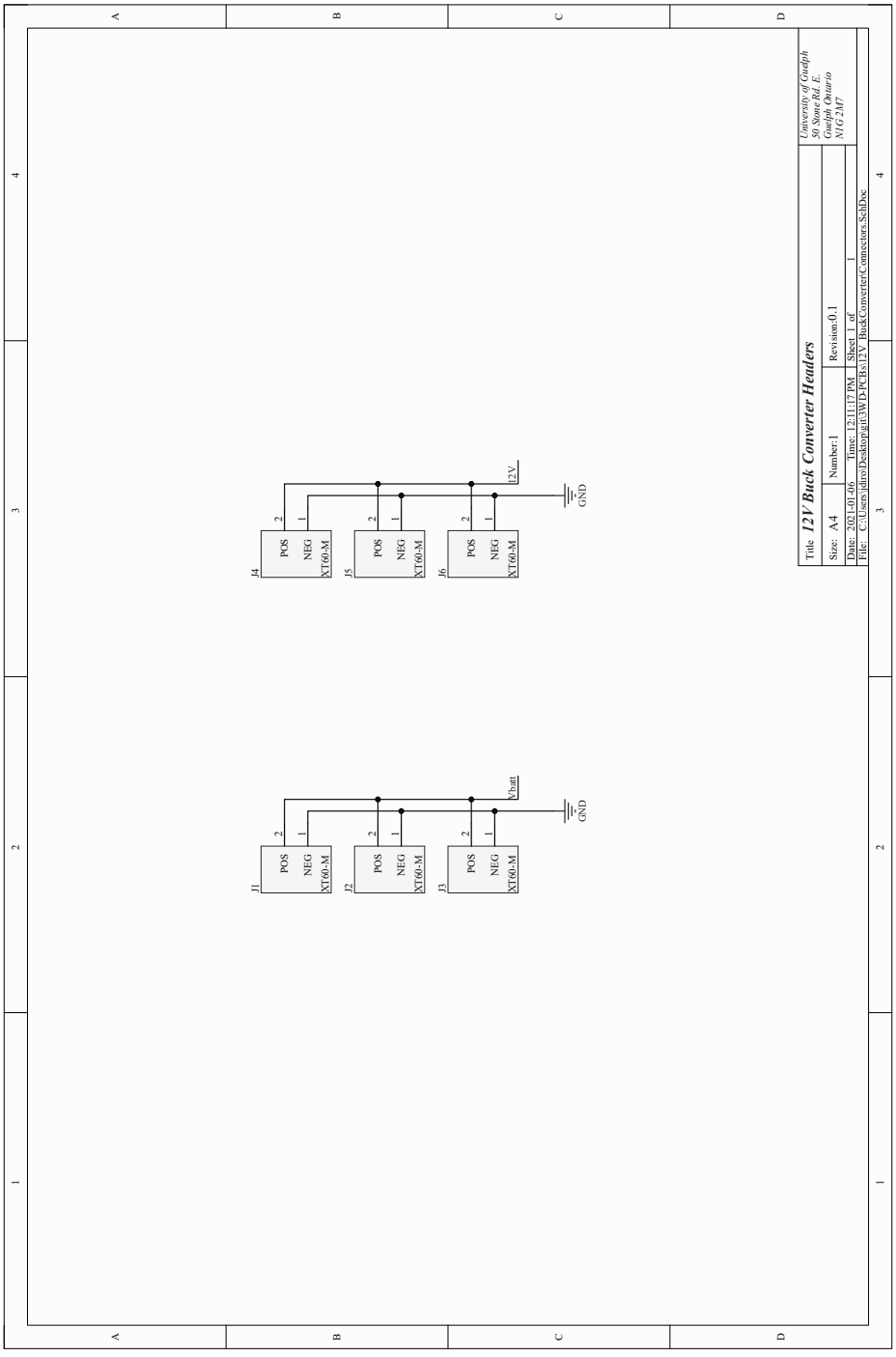

Figure A.2: 12V buck converter headers schematic.

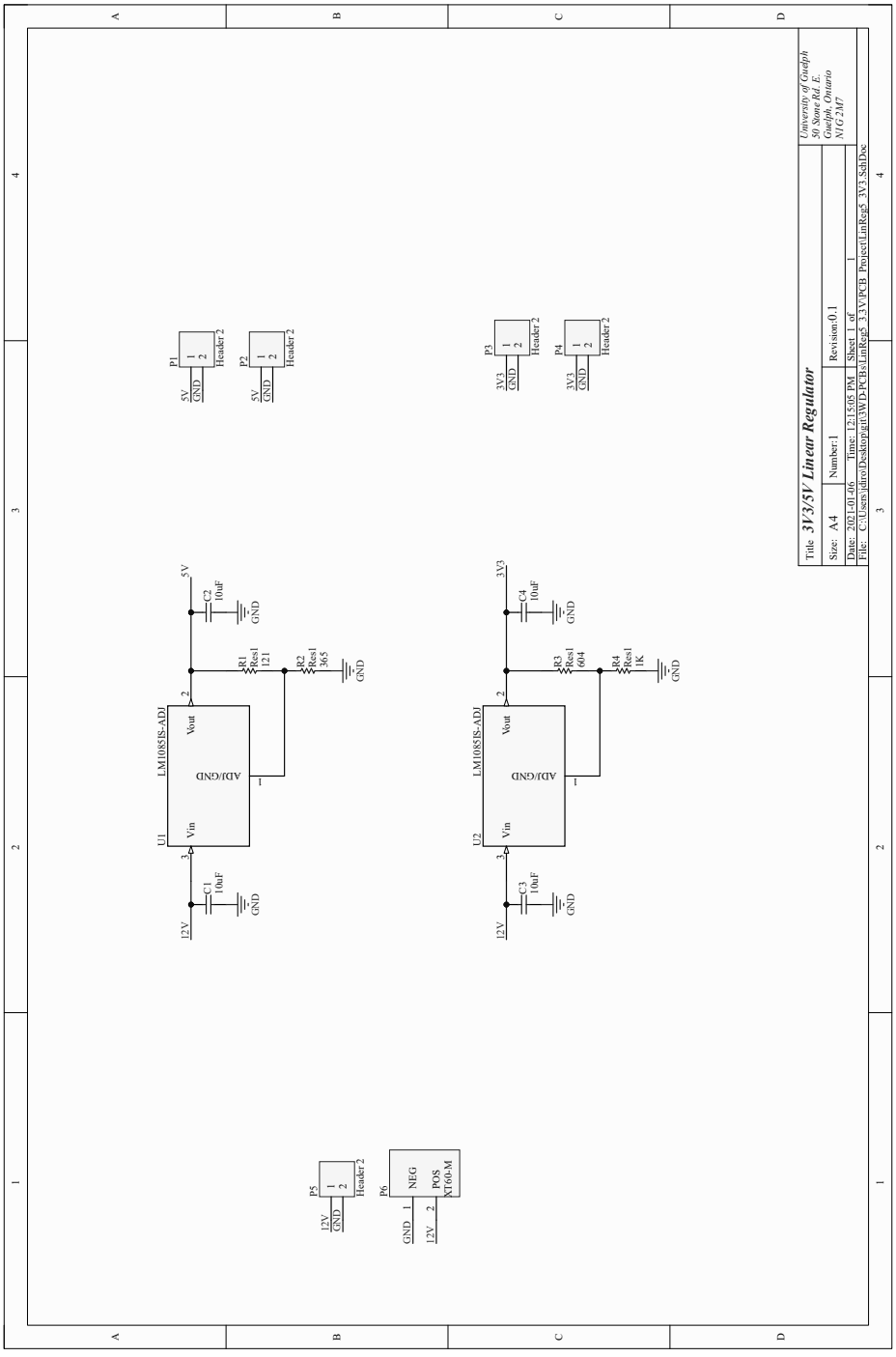

Figure A.3: 3.3/5V linear regulator schematic.

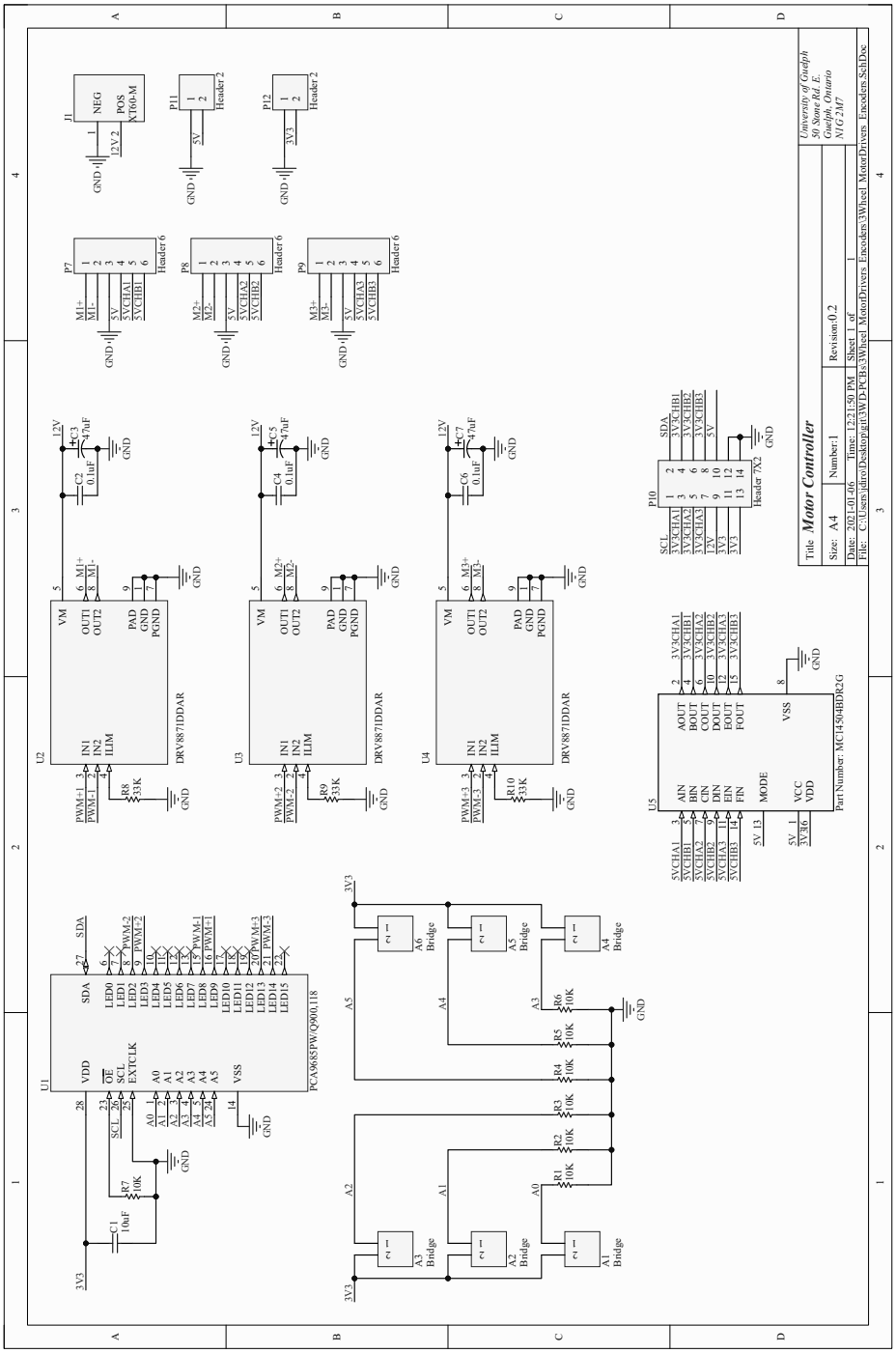

Figure A.4: Motor driver board schematic.

Appendix B

Chapter 4 Supplement

This appendix lists a few key scripts used for ARES. All scripts developed can be found at https://github.com/dyerbm/OmniWheel.

B.1 Sliding Mode Controller - Teensy Code

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <CircularBuffer.h>

#define MAX_PWM_FREQ 1600
#define FULL_ON_VAL 4096

Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);

char* output_string;

const int interruptPin_a_MSB = 3;
const int interruptPin_a_LSB = 2;
const int interruptPin_b_MSB = 7;
const int interruptPin_b_LSB = 6;
const int interruptPin_c_MSB = 5;
const int interruptPin_c_LSB = 4;
```

```
18
19 const int motor_a_p = 9;
20 const int motor_a_n = 8;
21 const int motor_b_p = 13;
22 const int motor_b_n = 14;
23 const int motor_c_p = 3;
24 const int motor_c_n = 2;
25
26 String echoString;
27
28 volatile double tics[3] = {0, 0, 0};
29 volatile double velocity[3] = {0, 0, 0};
30 char velocity_output[17];
31
32 int encoder_state[3];
33 int encoder_state_prev[3];
34
35 int incomingByte;
36
37 //Variables for the feedback lienarization controller
38 double v_r[3]={0, 0, 0}; //Linearized controller
39 double u_r[3]={0, 0, 0}; //Non-linear controller
40 double x_r[3]={0, 0, 0}; //Position of the robot
41 double x_r_desired[3]={0, 0, 0}; //desired position of robot
42 double xd_r_desired[3]={0,0,0}; //desired velocity of robot
43 const double wr=0.0508;//define wheel radius
44 const double rr=0.2632456;//define robot radius
45
46 const double lambda[3]={7,7,6}; //SMC lambda
47 const double K_r[3]={0.01,0.01,0.01}; //SMC K
48 double s[3] = {0,0,0}; //define sliding surface
49 double vshat[3] = {0,0,0}; //define measured slip
50
51 const int eint_r_length=100;
```

```
52 CircularBuffer<double,eint_r_length> e_r_x; //defines robot x
     error
53 double eint_r_x=0; //define integral error
54 double ed_r_x; //define derivative error
55 CircularBuffer<double,eint_r_length> e_r_y; //defines robot x
     error
56 double eint_r_y=0; //define integral error
57 double ed_r_y; //define derivative error
58 CircularBuffer<double,eint_r_length> e_r_t; //defines robot x
     error
59 double eint_r_t=0; //define integral error
60 double ed_r_t; //define derivative error
61
62 //Variables for the motor controller
63 const float T=20.0; //Desired time step in milliseconds
64 double omega_desired[3] = {0,0,0};
65 double u_m[3] = {0,0,0};
66
67 const double K_p = -10;
68 const double K_d = -0;
69 const double K_i = -30;
70
71 float time_m=0; //Current time (for the motors)
72 float time_previous_m=0; //Last time step for the motors
73
74 const int eint_m_length=100; //set bufer size based on desired
     integral loop
75 CircularBuffer<double,eint_m_length> e_m_a; //Initialize buffer
76 double eint_m_a=0;
77 double ed_m_a=0;
78 CircularBuffer<double,100> e_m_b;
79 double eint_m_b=0;
80 double ed_m_b=0;
81 CircularBuffer<double,100> e_m_c;
```

```
82 double eint_m_c=0;
83 double ed_m_c=0;
84
85
86 void setup() {
87   Serial1.begin(9600);
88   Serial.begin(115200); //Begin serial for XBee module
89   delay(3000);
90
91   pwm1.begin();
92   pwm1.setPWMFreq(MAX_PWM_FREQ);
93
94   //Initialize interupt pins for encoders
95   pinMode(interruptPin_a_MSB, INPUT);
96   pinMode(interruptPin_a_LSB, INPUT);
97   attachInterrupt(digitalPinToInterrupt(interruptPin_a_MSB),
        ENCODER_A, CHANGE);
98   attachInterrupt(digitalPinToInterrupt(interruptPin_a_LSB),
        ENCODER_A, CHANGE);
99   pinMode(interruptPin_b_MSB, INPUT);
100  pinMode(interruptPin_b_LSB, INPUT);
101  attachInterrupt(digitalPinToInterrupt(interruptPin_b_MSB),
        ENCODER_B, CHANGE);
102  attachInterrupt(digitalPinToInterrupt(interruptPin_b_LSB),
        ENCODER_B, CHANGE);
103  pinMode(interruptPin_c_MSB, INPUT);
104  pinMode(interruptPin_c_LSB, INPUT);
105  attachInterrupt(digitalPinToInterrupt(interruptPin_c_MSB),
        ENCODER_C, CHANGE);
106  attachInterrupt(digitalPinToInterrupt(interruptPin_c_LSB),
        ENCODER_C, CHANGE);
107
108  //set all motors to not move
109  pwm1.setPWM(motor_a_p, 0, FULL_ON_VAL);
```

```
110   pwm1.setPWM(motor_a_n, 0, FULL_ON_VAL);
111   pwm1.setPWM(motor_b_p, 0, FULL_ON_VAL);
112   pwm1.setPWM(motor_b_n, 0, FULL_ON_VAL);
113   pwm1.setPWM(motor_c_p, 0, FULL_ON_VAL);
114   pwm1.setPWM(motor_c_n, 0, FULL_ON_VAL);
115   delay(1000);
116
117   for (int i=0;i<=eint_m_length;i++){
118     e_m_a.unshift(0);
119     e_m_b.unshift(0);
120     e_m_c.unshift(0);
121     e_r_x.unshift(0);
122     e_r_y.unshift(0);
123     e_r_t.unshift(0);
124   }
125
126   delay(1000); //make sure motors stop turning so the robot
        starts at origin
127 }
128
129 int start_time = millis();
130 String cString;
131 int ind1,ind2,ind3,ind4,ind5,ind6;
132
133 void loop() {
134   time_m=millis();
135
136   if (Serial1.available()>0) {
137 //    digitalWrite(ledPin, HIGH); //indicate serial is being read
138     char c = Serial1.read();
139
140     if (c=='*'){
141       //Serial.println(cString);
142       ind1 = cString.indexOf(',');  //finds location of first ','
```

```
143       x_r_desired[0] = cString.substring(0, ind1).toFloat();   //
              captures first data String
144       ind2 = cString.indexOf(',', ind1+1 ); //finds location of
              second ','
145       x_r_desired[1] = cString.substring(ind1 + 1,ind2).toFloat()
              ; //captures second data String
146       ind3 = cString.indexOf(',', ind2 +1  ); //finds location of
               third ','
147       x_r_desired[2] = cString.substring(ind2+1,ind3).toFloat();
              //captures third data String
148
149       cString="";
150     }
151
152     else {
153       cString += c; //makes the string controlString
154     }
155   }
156
157   // Main control loop, should be changed to run on a watchdog
       interupt
158   if (time_m-time_previous_m>=T) {
159
160     CALC_VELOCITY(time_m-time_previous_m); //calculate each wheel
          velocity
161
162     //-----------Calculate Sliding Mode Control--------------//
163     //Calculate current position
164
165     x_r[0] = x_r[0] + ((2.0/3.0*sin(x_r[2]))*velocity[0]+(cos(x_r
            [2])/sqrt(3.0)-sin(x_r[2])/3.0)*velocity[1]+(-cos(x_r[2])/
            sqrt(3.0)-sin(x_r[2])/3.0)*velocity[2])*(time_m-
            time_previous_m)/1000.0*wr*1.0196; //calculate new
            position, use scaling factor
```

```
166     x_r[1] = x_r[1] + ((-2.0/3.0*cos(x_r[2]))*velocity[0]+(sin(
           x_r[2])/sqrt(3.0)+cos(x_r[2])/3.0)*velocity[1]+(-sin(x_r
           [2])/sqrt(3.0)+cos(x_r[2])/3.0)*velocity[2])*(time_m-
           time_previous_m)/1000.0*wr*1.0254; //calculate new
           position, use scaling factor
167     x_r[2] = x_r[2] + (-1./(3.*rr)*(velocity[0]+velocity[1]+
           velocity[2]))*(time_m-time_previous_m)/1000.*wr*1.0023; //
           calculate new position, use scaling factor
168
169     e_r_x.unshift(x_r[0]-x_r_desired[0]);//calculate new error
170     e_r_y.unshift(x_r[1]-x_r_desired[1]);
171     e_r_t.unshift(x_r[2]-x_r_desired[2]);
172     eint_r_x = eint_r_x+(e_r_x[0]-e_r_x[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;//recalculate integral
           error
173     eint_r_y = eint_r_y+(e_r_y[0]-e_r_y[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;
174     eint_r_t = eint_r_t+(e_r_t[0]-e_r_t[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;
175     ed_r_x = (e_r_x[0]-e_r_x[1])/(T/1000.0); //recalculate
           derivative error
176     ed_r_y = (e_r_y[0]-e_r_y[1])/(T/1000.0);
177     ed_r_t = (e_r_t[0]-e_r_t[1])/(T/1000.0);
178
179     //PUT UR CONTROLLER HERE!
180
181     s[0]=e_r_x[0]+lambda[0]*eint_r_x; //calculate sliding surface
182     s[1]=e_r_y[0]+lambda[1]*eint_r_y;
183     s[2]=e_r_t[0]+lambda[2]*eint_r_t;
184
185     omega_desired[0]=1/wr*(sin(x_r[2])*(-lambda[0]*e_r_x[0]+
           xd_r_desired[0]-K_r[0]*sgn(s[0]))-cos(x_r[2])*(-lambda[1]*
           e_r_y[0]+xd_r_desired[1]-K_r[1]*sgn(s[1]))-rr*(-lambda[2]*
           e_r_t[0]+xd_r_desired[2]-K_r[2]*sgn(s[2])))-vshat[0];
```

```
186     omega_desired[1]=1/wr*((sqrt(3.)/2.*cos(x_r[2])-sin(x_r[2])
           /2.)*(-lambda[0]*e_r_x[0]+xd_r_desired[0]-K_r[0]*sgn(s[0])
           )+(sqrt(3.)/2.*sin(x_r[2])+cos(x_r[2])/2.)*(-lambda[1]*
           e_r_y[0]+xd_r_desired[1]-K_r[1]*sgn(s[1]))-rr*(-lambda[2]*
           e_r_t[0]+xd_r_desired[2]-K_r[2]*sgn(s[2])))-vshat[1];
187     omega_desired[2]=1/wr*((-sqrt(3.)/2.*cos(x_r[2])-sin(x_r[2])
           /2.)*(-lambda[0]*e_r_x[0]+xd_r_desired[0]-K_r[0]*sgn(s[0])
           )+(-sqrt(3.)/2.*sin(x_r[2])+cos(x_r[2])/2.)*(-lambda[1]*
           e_r_y[0]+xd_r_desired[1]-K_r[1]*sgn(s[1]))-rr*(-lambda[2]*
           e_r_t[0]+xd_r_desired[2]-K_r[2]*sgn(s[2])))-vshat[2];
188
189
190     if (abs(omega_desired[0])>4 || abs(omega_desired[1])>4 || abs
           (omega_desired[2])>4) {
191       float quickcounter = omega_desired[0];
192       for (int i=1; i<3;i++){
193         if (abs(quickcounter)<abs(omega_desired[i])) (
              quickcounter=omega_desired[i]);
194       }
195       omega_desired[0]=omega_desired[0]/abs(quickcounter)*4;
196       omega_desired[1]=omega_desired[1]/abs(quickcounter)*4;
197       omega_desired[2]=omega_desired[2]/abs(quickcounter)*4;
198     }
199
200     //Calculate Motor Controllers
201     e_m_a.unshift(velocity[0]-omega_desired[0]);
202     eint_m_a = eint_m_a+(e_m_a[0]-e_m_a[eint_m_length-1])*(T
           /1000)/eint_m_length;
203     ed_m_a = (e_m_a[0]-e_m_a[1])/(T/1000);
204     u_m[0] = K_p*e_m_a[0]+K_d*ed_m_a+K_i*eint_m_a; //calculate
           the controller in PWM
205
206     e_m_b.unshift(velocity[1]-omega_desired[1]);
```

```
207     eint_m_b = eint_m_b+(e_m_b[0]-e_m_b[eint_m_length-1])*(T
           /1000)/eint_m_length;
208     ed_m_b = (e_m_b[0]-e_m_b[1])/(T/1000);
209     u_m[1] = K_p*e_m_b[0]+K_d*ed_m_b+K_i*eint_m_b; //calculate
           the controller in PWM
210
211     e_m_c.unshift(velocity[2]-omega_desired[2]);
212     eint_m_c = eint_m_c+(e_m_c[0]-e_m_c[eint_m_length-1])*(T
           /1000)/eint_m_length;
213     ed_m_c = (e_m_c[0]-e_m_c[1])/(T/1000);
214     u_m[2] = K_p*e_m_c[0]+K_d*ed_m_a+K_i*eint_m_a; //calculate
           the controller in PWM
215
216     if (abs(u_m[0])>12 || abs(u_m[1])>12 || abs(u_m[2])>12) {
217       float quickcounter = u_m[0];
218       for (int i=1; i<3;i++){
219         if (abs(quickcounter)<abs(u_m[i])) (quickcounter=u_m[i]);
220       }
221       u_m[0]=u_m[0]/abs(quickcounter)*12;
222       u_m[1]=u_m[1]/abs(quickcounter)*12;
223       u_m[2]=u_m[2]/abs(quickcounter)*12;
224       //Serial.println(millis());
225     }
226
227     //u_m[0]=8;u_m[1]=-4;u_m[2]=-4; //This forces a spinning
           controller
228
229     //-----Set each motor accordingly-----//
230     if (abs(e_r_x[0])>0.002 || abs(e_r_y[0])>0.002 || abs(e_r_t
           [0])>0.05){ //Make sure the robot doesn't move if it's
           within 2 mm and 5 degrees
231       if (u_m[0]>0) {
232         pwm1.setPWM(motor_a_n, 0, (int) (abs(u_m[0])/12*4096));
233         pwm1.setPWM(motor_a_p, 0, 0);
```

```
234       }
235       else {
236         pwm1.setPWM(motor_a_n, 0, 0);
237         pwm1.setPWM(motor_a_p, 0, (int) (abs(u_m[0])/12*4096));
238       }
239       if (u_m[1]>0) {
240         pwm1.setPWM(motor_b_n, 0, (int) (abs(u_m[1])/12*4096));
241         pwm1.setPWM(motor_b_p, 0, 0);
242       }
243       else {
244         pwm1.setPWM(motor_b_n, 0, 0);
245         pwm1.setPWM(motor_b_p, 0, (int) (abs(u_m[1])/12*4096));
246       }
247       if (u_m[2]>0) {
248         pwm1.setPWM(motor_c_n, 0, (int) (abs(u_m[2])/12*4096));
249         pwm1.setPWM(motor_c_p, 0, 0);
250       }
251       else {
252         pwm1.setPWM(motor_c_n, 0, 0);
253         pwm1.setPWM(motor_c_p, 0, (int) (abs(u_m[2])/12*4096));
254       }
255     }
256     else { //if robot is close enough, stop moving the motors (
          will help stop the noise)
257       pwm1.setPWM(motor_a_n, 0, 0);
258       pwm1.setPWM(motor_a_p, 0, 0);
259       pwm1.setPWM(motor_b_n, 0, 0);
260       pwm1.setPWM(motor_b_p, 0, 0);
261       pwm1.setPWM(motor_c_n, 0, 0);
262       pwm1.setPWM(motor_c_p, 0, 0);
263     }
264
265
266     time_previous_m=time_m; //set previous time to current time
```

```
267
268
269   }
270 }
271
272 static inline double sgn(double val) {
273  if (val < 0) return -1.;
274  if (val==0) return 0.;
275  return 1.;
276 }
277
278 // These encoder functions were origionally written by Trevor
      Smith (minor changes made)
279 // Encoder A
280 void ENCODER_A() {
281   noInterrupts();
282   UPDATE_STATES_A(); //updates encoder values
283   if (encoder_state[0] != encoder_state_prev[0]) { //update tics
        if change detected
284     if ( encoder_state[0]==3&&encoder_state_prev[0]==1 ||
          encoder_state[0]==2&&encoder_state_prev[0]==3 ||
          encoder_state[0]==0&&encoder_state_prev[0]==2 ||
          encoder_state[0]==1&&encoder_state_prev[0]==0 ) {
285       tics[0] = tics[0] + 1;
286     } else {
287       tics[0] = tics[0] - 1;
288     }
289   }
290   interrupts();
291 }
292
293 void UPDATE_STATES_A() {
294   encoder_state_prev[0] = encoder_state[0]; // Update previous
        int
```

```
295   encoder_state[0] = digitalRead(interruptPin_a_MSB) * 2 +
          digitalRead(interruptPin_a_LSB);
296 }
297
298 // Encoder B
299 void ENCODER_B() {
300   noInterrupts();
301   UPDATE_STATES_B(); //updates encoder values
302   if (encoder_state[1] != encoder_state_prev[1]) { //update tics
          if change detected
303     if ( encoder_state[1]==3&&encoder_state_prev[1]==1 ||
            encoder_state[1]==2&&encoder_state_prev[1]==3 ||
            encoder_state[1]==0&&encoder_state_prev[1]==2 ||
            encoder_state[1]==1&&encoder_state_prev[1]==0 ) {
304       tics[1] = tics[1] + 1;
305     } else {
306       tics[1] = tics[1] - 1;
307     }
308   }
309   interrupts();
310 }
311
312 void UPDATE_STATES_B() {
313   encoder_state_prev[1] = encoder_state[1]; // Update previous
          int
314   encoder_state[1] = digitalRead(interruptPin_b_MSB) * 2 +
          digitalRead(interruptPin_b_LSB);
315 }
316
317 // Encoder C
318 void ENCODER_C() {
319   noInterrupts();
320   UPDATE_STATES_C(); //updates encoder values
```

```
321   if (encoder_state[2] != encoder_state_prev[2]) { //update tics
         if change detected
322     if ( encoder_state[2]==3&&encoder_state_prev[2]==1 ||
           encoder_state[2]==2&&encoder_state_prev[2]==3 ||
           encoder_state[2]==0&&encoder_state_prev[2]==2 ||
           encoder_state[2]==1&&encoder_state_prev[2]==0 ) {
323       tics[2] = tics[2] + 1;
324     } else {
325       tics[2] = tics[2] - 1;
326     }
327   }
328   interrupts();
329 }
330
331 void UPDATE_STATES_C() {
332   encoder_state_prev[2] = encoder_state[2]; // Update previous
         int
333   encoder_state[2] = digitalRead(interruptPin_c_MSB) * 2 +
         digitalRead(interruptPin_c_LSB);
334 }
335
336 void CALC_VELOCITY(float timestep) {
337   noInterrupts();
338   velocity[0] = tics[0] / ((timestep)/1000) / 10000*2*3.14156; //
         give time in tics per time period
339   velocity[1] = tics[1] / ((timestep)/1000) / 10000*2*3.14156;
340   velocity[2] = tics[2] / ((timestep)/1000) / 10000*2*3.14156;
341   tics[0] = 0;
342   tics[1] = 0;
343   tics[2] = 0;
344   interrupts();
345 }
```

B.2 Feedback Linearization Controller - Teensy Code

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <CircularBuffer.h>

#define MAX_PWM_FREQ 1600
#define FULL_ON_VAL 4096

Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);

char* output_string;

const int interruptPin_a_MSB = 3;
const int interruptPin_a_LSB = 2;
const int interruptPin_b_MSB = 7;
const int interruptPin_b_LSB = 6;
const int interruptPin_c_MSB = 5;
const int interruptPin_c_LSB = 4;

const int motor_a_p = 9;
const int motor_a_n = 8;
const int motor_b_p = 13;
const int motor_b_n = 14;
const int motor_c_p = 3;
const int motor_c_n = 2;

String echoString;

volatile double tics[3] = {0, 0, 0};
volatile double velocity[3] = {0, 0, 0};
char velocity_output[17];

```

```
32 int encoder_state[3];
33 int encoder_state_prev[3];
34
35 int incomingByte;
36
37 //Variables for the feedback lienarization controller
38 double v_r[3]={0, 0, 0}; //Linearized controller
39 double u_r[3]={0, 0, 0}; //Non-linear controller
40 double x_r[3]={0,0,0}; //Position of the robot
41 double x_r_desired[3]={0,0,0}; //desired position of robot
42 const double wr=0.0508;//define wheel radius
43 const double rr=0.2632456;//define robot radius
44
45 const double K_p_r[3]={-20,-20,-6}; //proportional gains
46 const double K_d_r[3]={0,0,0}; //derivative gain
47 const double K_i_r[3]={-50,-50,-30}; //integral gain (in sim was
     -2000)
48
49 const int eint_r_length=100;
50 CircularBuffer<double,eint_r_length> e_r_x; //defines robot x
     error
51 double eint_r_x=0; //define integral error
52 double ed_r_x; //define derivative error
53 CircularBuffer<double,eint_r_length> e_r_y; //defines robot x
     error
54 double eint_r_y=0; //define integral error
55 double ed_r_y; //define derivative error
56 CircularBuffer<double,eint_r_length> e_r_t; //defines robot x
     error
57 double eint_r_t=0; //define integral error
58 double ed_r_t; //define derivative error
59
60 //Variables for the motor controller
61 const float T=20.0; //Desired time step in milliseconds
```

```
62 double omega_desired[3] = {0,0,0};
63 double u_m[3] = {0,0,0};
64
65 const double K_p = -10;
66 const double K_d = -0;
67 const double K_i = -30;
68
69 float time_m=0; //Current time (for the motors)
70 float time_previous_m=0; //Last time step for the motors
71
72 const int eint_m_length=100; //set bufer size based on desired
     integral loop
73 CircularBuffer<double,eint_m_length> e_m_a; //Initialize buffer
74 double eint_m_a=0;
75 double ed_m_a=0;
76 CircularBuffer<double,100> e_m_b;
77 double eint_m_b=0;
78 double ed_m_b=0;
79 CircularBuffer<double,100> e_m_c;
80 double eint_m_c=0;
81 double ed_m_c=0;
82
83
84 void setup() {
85   Serial.begin(115200);
86   Serial1.begin(9600);
87   delay(3000);
88
89   pwm1.begin();
90   pwm1.setPWMFreq(MAX_PWM_FREQ);
91
92   //Initialize interupt pins for encoders
93   pinMode(interruptPin_a_MSB, INPUT);
94   pinMode(interruptPin_a_LSB, INPUT);
```

```
95  attachInterrupt(digitalPinToInterrupt(interruptPin_a_MSB),
       ENCODER_A, CHANGE);
96  attachInterrupt(digitalPinToInterrupt(interruptPin_a_LSB),
       ENCODER_A, CHANGE);
97  pinMode(interruptPin_b_MSB, INPUT);
98  pinMode(interruptPin_b_LSB, INPUT);
99  attachInterrupt(digitalPinToInterrupt(interruptPin_b_MSB),
       ENCODER_B, CHANGE);
100 attachInterrupt(digitalPinToInterrupt(interruptPin_b_LSB),
       ENCODER_B, CHANGE);
101 pinMode(interruptPin_c_MSB, INPUT);
102 pinMode(interruptPin_c_LSB, INPUT);
103 attachInterrupt(digitalPinToInterrupt(interruptPin_c_MSB),
       ENCODER_C, CHANGE);
104 attachInterrupt(digitalPinToInterrupt(interruptPin_c_LSB),
       ENCODER_C, CHANGE);
105
106 //set all motors to not move
107 pwm1.setPWM(motor_a_p, 0, FULL_ON_VAL);
108 pwm1.setPWM(motor_a_n, 0, FULL_ON_VAL);
109 pwm1.setPWM(motor_b_p, 0, FULL_ON_VAL);
110 pwm1.setPWM(motor_b_n, 0, FULL_ON_VAL);
111 pwm1.setPWM(motor_c_p, 0, FULL_ON_VAL);
112 pwm1.setPWM(motor_c_n, 0, FULL_ON_VAL);
113 delay(1000);
114
115 for (int i=0;i<=eint_m_length;i++){
116   e_m_a.unshift(0);
117   e_m_b.unshift(0);
118   e_m_c.unshift(0);
119   e_r_x.unshift(0);
120   e_r_y.unshift(0);
121   e_r_t.unshift(0);
122 }
```

```
123
124   delay(1000); //make sure motors stop turning so the robot
          starts at origin
125 }
126
127 int start_time = millis();
128 String cString;
129 int ind1,ind2,ind3;
130
131 void loop() {
132   time_m=millis();
133
134   if (Serial1.available()>0) {
135 //    digitalWrite(ledPin, HIGH); //indicate serial is being read
136     char c = Serial1.read();
137
138     if (c=='*'){
139       Serial.println(cString);
140       ind1 = cString.indexOf(',');  //finds location of first ','
141       x_r_desired[0] = cString.substring(0, ind1).toFloat();   //
            captures first data String
142       ind2 = cString.indexOf(',', ind1+1 ); //finds location of
            second ','
143       x_r_desired[1] = cString.substring(ind1 + 1,ind2).toFloat()
            ; //captures second data String
144       ind3 = cString.indexOf(',', ind2 +1  ); //finds location of
             second ','
145       x_r_desired[2] = cString.substring(ind2+1,ind3).toFloat();
            //captures third data String
146
147
148       cString="";
149     }
150
```

```
151     else {
152       cString += c; //makes the string controlString
153     }
154   }
155 //  digitalWrite(ledPin, LOW);
156
157   if (time_m-time_previous_m>=T) {
158
159
160     CALC_VELOCITY(time_m-time_previous_m); //calculate each wheel
          velocity
161
162
163     //-----------Calculate Feedback linearization--------------//
164     //Calculate current position
165
166
167     x_r[0] = x_r[0] + ((2.0/3.0*sin(x_r[2]))*velocity[0]+(cos(x_r
          [2])/sqrt(3.0)-sin(x_r[2])/3.0)*velocity[1]+(-cos(x_r[2])/
          sqrt(3.0)-sin(x_r[2])/3.0)*velocity[2])*(time_m-
          time_previous_m)/1000.0*wr*1.0196; //calculate new
          position, use scaling factor
168     x_r[1] = x_r[1] + ((-2.0/3.0*cos(x_r[2]))*velocity[0]+(sin(
          x_r[2])/sqrt(3.0)+cos(x_r[2])/3.0)*velocity[1]+(-sin(x_r
          [2])/sqrt(3.0)+cos(x_r[2])/3.0)*velocity[2])*(time_m-
          time_previous_m)/1000.0*wr*1.0254; //calculate new
          position, use scaling factor
169     x_r[2] = x_r[2] + (-1./(3.*rr)*(velocity[0]+velocity[1]+
          velocity[2]))*(time_m-time_previous_m)/1000.*wr*1.0023; //
          calculate new position, use scaling factor
170
171     e_r_x.unshift(x_r[0]-x_r_desired[0]);//calculate new error
172     e_r_y.unshift(x_r[1]-x_r_desired[1]);
173     e_r_t.unshift(x_r[2]-x_r_desired[2]);
```

```
174     eint_r_x = eint_r_x+(e_r_x[0]-e_r_x[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;//recalculate integral
           error
175     eint_r_y = eint_r_y+(e_r_y[0]-e_r_y[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;
176     eint_r_t = eint_r_t+(e_r_t[0]-e_r_t[eint_r_length-1])*(T
           /1000.0)/(double)eint_r_length;
177     ed_r_x = (e_r_x[0]-e_r_x[1])/(T/1000.0); //recalculate
           derivative error
178     ed_r_y = (e_r_y[0]-e_r_y[1])/(T/1000.0);
179     ed_r_t = (e_r_t[0]-e_r_t[1])/(T/1000.0);
180
181     v_r[0]= K_p_r[0]*e_r_x[0]+K_d_r[0]*ed_r_x+K_i_r[0]*eint_r_x;
           //calculate linear controller
182     v_r[1]= K_p_r[1]*e_r_y[0]+K_d_r[1]*ed_r_y+K_i_r[1]*eint_r_y;
183     v_r[2]= K_p_r[2]*e_r_t[0]+K_d_r[2]*ed_r_t+K_i_r[2]*eint_r_t;
184
185     omega_desired[0] = (sin(x_r[2])*v_r[0]-cos(x_r[2])*v_r[1]-rr*
           v_r[2])*(time_m-time_previous_m)*wr; //x position
186     omega_desired[1] = ((sqrt(3)/2*cos(x_r[2])-sin(x_r[2])/2)*v_r
           [0]+(sqrt(3)/2*sin(x_r[2])+cos(x_r[2])/2)*v_r[1]+(-rr)*v_r
           [2])*(time_m-time_previous_m)*wr;
187     omega_desired[2] = ((-sqrt(3)/2*cos(x_r[2])-sin(x_r[2])/2)*
           v_r[0]+(-sqrt(3)/2*sin(x_r[2])+cos(x_r[2])/2)*v_r[1]+(-rr)
           *v_r[2])*(time_m-time_previous_m)*wr;
188
189     if (abs(omega_desired[0])>4 || abs(omega_desired[1])>4 || abs
           (omega_desired[2])>4) {
190       float quickcounter = omega_desired[0];
191       for (int i=1; i<3;i++){
192         if (abs(quickcounter)<abs(omega_desired[i])) (
              quickcounter=omega_desired[i]);
193       }
194       omega_desired[0]=omega_desired[0]/abs(quickcounter)*4;
```

```
195       omega_desired[1]=omega_desired[1]/abs(quickcounter)*4;
196       omega_desired[2]=omega_desired[2]/abs(quickcounter)*4;
197     }
198
199     //Calculate Motor Controllers
200     e_m_a.unshift(velocity[0]-omega_desired[0]);
201     eint_m_a = eint_m_a+(e_m_a[0]-e_m_a[eint_m_length-1])*(T
          /1000)/eint_m_length;
202     ed_m_a = (e_m_a[0]-e_m_a[1])/(T/1000);
203     u_m[0] = K_p*e_m_a[0]+K_d*ed_m_a+K_i*eint_m_a; //calculate
          the controller in PWM
204
205     e_m_b.unshift(velocity[1]-omega_desired[1]);
206     eint_m_b = eint_m_b+(e_m_b[0]-e_m_b[eint_m_length-1])*(T
          /1000)/eint_m_length;
207     ed_m_b = (e_m_b[0]-e_m_b[1])/(T/1000);
208     u_m[1] = K_p*e_m_b[0]+K_d*ed_m_b+K_i*eint_m_b; //calculate
          the controller in PWM
209
210     e_m_c.unshift(velocity[2]-omega_desired[2]);
211     eint_m_c = eint_m_c+(e_m_c[0]-e_m_c[eint_m_length-1])*(T
          /1000)/eint_m_length;
212     ed_m_c = (e_m_c[0]-e_m_c[1])/(T/1000);
213     u_m[2] = K_p*e_m_c[0]+K_d*ed_m_a+K_i*eint_m_a; //calculate
          the controller in PWM
214
215     if (abs(u_m[0])>12 || abs(u_m[1])>12 || abs(u_m[2])>12) {
216       float quickcounter = u_m[0];
217       for (int i=1; i<3;i++){
218         if (abs(quickcounter)<abs(u_m[i])) (quickcounter=u_m[i]);
219       }
220       u_m[0]=u_m[0]/abs(quickcounter)*12;
221       u_m[1]=u_m[1]/abs(quickcounter)*12;
222       u_m[2]=u_m[2]/abs(quickcounter)*12;
```

```
223       //Serial.println(millis());
224     }
225
226     //u_m[0]=8;u_m[1]=-4;u_m[2]=-4; //This forces a spinning
          controller
227
228     //-----Set each motor accordingly-----//
229     if (u_m[0]>0) {
230       pwm1.setPWM(motor_a_n, 0, (int) (abs(u_m[0])/12*4096));
231       pwm1.setPWM(motor_a_p, 0, 0);
232     }
233     else {
234       pwm1.setPWM(motor_a_n, 0, 0);
235       pwm1.setPWM(motor_a_p, 0, (int) (abs(u_m[0])/12*4096));
236     }
237     if (u_m[1]>0) {
238       pwm1.setPWM(motor_b_n, 0, (int) (abs(u_m[1])/12*4096));
239       pwm1.setPWM(motor_b_p, 0, 0);
240     }
241     else {
242       pwm1.setPWM(motor_b_n, 0, 0);
243       pwm1.setPWM(motor_b_p, 0, (int) (abs(u_m[1])/12*4096));
244     }
245     if (u_m[2]>0) {
246       pwm1.setPWM(motor_c_n, 0, (int) (abs(u_m[2])/12*4096));
247       pwm1.setPWM(motor_c_p, 0, 0);
248     }
249     else {
250       pwm1.setPWM(motor_c_n, 0, 0);
251       pwm1.setPWM(motor_c_p, 0, (int) (abs(u_m[2])/12*4096));
252     }
253
254
255     time_previous_m=time_m; //set previous time to current time
```

```
256
257
258     //sprintf(output_string, "Error: %f\tController: %d",e_m_a
           [0],u_m[0]); //Send out error string (leave commented
           unless you have the bandwidth)
259     //Serial.println(output_string);
260     Serial.print(x_r[0],5);
261     Serial.print(",");
262     Serial.print(x_r[1],5);
263     Serial.print(",");
264     Serial.print(x_r[2],5);
265     Serial.print("\n");
266   }
267
268 }
269
270 // Encoder A
271 void ENCODER_A() {
272   noInterrupts();
273   UPDATE_STATES_A(); //updates encoder values
274   if (encoder_state[0] != encoder_state_prev[0]) { //update tics
        if change detected
275     if ( encoder_state[0]==3&&encoder_state_prev[0]==1 ||
           encoder_state[0]==2&&encoder_state_prev[0]==3 ||
           encoder_state[0]==0&&encoder_state_prev[0]==2 ||
           encoder_state[0]==1&&encoder_state_prev[0]==0 ) {
276       tics[0] = tics[0] + 1;
277     } else {
278       tics[0] = tics[0] - 1;
279     }
280   }
281   interrupts();
282 }
283
```

```
284 void UPDATE_STATES_A() {
285   encoder_state_prev[0] = encoder_state[0]; // Update previous
        int
286   encoder_state[0] = digitalRead(interruptPin_a_MSB) * 2 +
        digitalRead(interruptPin_a_LSB);
287 }
288
289 // Encoder B
290 void ENCODER_B() {
291   noInterrupts();
292   UPDATE_STATES_B(); //updates encoder values
293   if (encoder_state[1] != encoder_state_prev[1]) { //update tics
        if change detected
294     if ( encoder_state[1]==3&&encoder_state_prev[1]==1 ||
          encoder_state[1]==2&&encoder_state_prev[1]==3 ||
          encoder_state[1]==0&&encoder_state_prev[1]==2 ||
          encoder_state[1]==1&&encoder_state_prev[1]==0 ) {
295       tics[1] = tics[1] + 1;
296     } else {
297       tics[1] = tics[1] - 1;
298     }
299   }
300   interrupts();
301 }
302
303 void UPDATE_STATES_B() {
304   encoder_state_prev[1] = encoder_state[1]; // Update previous
        int
305   encoder_state[1] = digitalRead(interruptPin_b_MSB) * 2 +
        digitalRead(interruptPin_b_LSB);
306 }
307
308 // Encoder C
309 void ENCODER_C() {
```

```
310   noInterrupts();
311   UPDATE_STATES_C(); //updates encoder values
312   if (encoder_state[2] != encoder_state_prev[2]) { //update tics
         if change detected
313     if ( encoder_state[2]==3&&encoder_state_prev[2]==1 ||
           encoder_state[2]==2&&encoder_state_prev[2]==3 ||
           encoder_state[2]==0&&encoder_state_prev[2]==2 ||
           encoder_state[2]==1&&encoder_state_prev[2]==0 ) {
314       tics[2] = tics[2] + 1;
315     } else {
316       tics[2] = tics[2] - 1;
317     }
318   }
319   interrupts();
320 }
321
322 void UPDATE_STATES_C() {
323   encoder_state_prev[2] = encoder_state[2]; // Update previous
         int
324   encoder_state[2] = digitalRead(interruptPin_c_MSB) * 2 +
         digitalRead(interruptPin_c_LSB);
325 }
326
327 void CALC_VELOCITY(float timestep) {
328   noInterrupts();
329   velocity[0] = tics[0] / ((timestep)/1000) / 10000*2*3.14156; //
         give time in tics per time period
330   velocity[1] = tics[1] / ((timestep)/1000) / 10000*2*3.14156;
331   velocity[2] = tics[2] / ((timestep)/1000) / 10000*2*3.14156;
332   tics[0] = 0;
333   tics[1] = 0;
334   tics[2] = 0;
335   interrupts();
336 }
```

B.3 Vicon Data Collection - Matlab Code

```
% The majority of this code was provided by Amit Patel
% It has been modified to run a different robot at a different
% frequency for custom trajectories


timenow=0.02;
timeglobal=0;
tglobal=tic;
desired=[0,0,0];

marker_1_lost = false;
marker_2_lost = false;
marker_3_lost = false;
marker_4_lost = false;
marker_5_lost = false;
marker_6_lost = false;
marker_7_lost = false;
marker_8_lost = false;

matcounter = 1; % Starting row for output matrix
max_operation = 120; % Time to record (max time robot will move
   if sending signal)
matrixsize = max_operation * 50 + 50; % Based on the time for
   operation, will wait 1 second after robot stops to end
   recording

Sheet1Mat = zeros(matrixsize,29);

% Below are the headings for the files commented, feel
%Raw_Headings = ['GlobalTime', 'Time', 'M1X', 'M1Y', 'M1Z', 'M2X
   ', 'M2Y', 'M2Z', 'M3X', 'M3Y', 'M3Z', 'M4X', 'M4Y', 'M4Z', '
```

```
    M5X', 'M5Y', 'M5Z','M6X', 'M6Y', 'M6Z','M7X', 'M7Y', 'M7Z','
    M8X', 'M8Y', 'M8Z'];

%% select serial port.
delete(instrfind);
port = 'COM7'; % Replace with whatever the USB serial bus from
    the XBee module is on (was 7)
serialPortObj = serial(port, 'BaudRate', 9600);
fopen(serialPortObj);

%% Get name of notebook from user
prompt={'Please enter the name of the desired notebook'};
title='Excel notebook name';
notebook_name = inputdlg(prompt,title);
notebook_name_raw = strcat(notebook_name{1}, '_raw', '.xlsx');

%% CONNECT TO DATA STREAM
% Load the SDK
Client.LoadViconDataStreamSDK();
fprintf( 'Vicon Data Stream SDK loaded\n' );

% Connect to a server
HostName = 'localhost:801';
MyClient = Client(); % Client obj
fprintf( 'Connecting to %s ...\n', HostName );
while ~MyClient.IsConnected().Connected
  MyClient.Connect( HostName );
end

% Enable some different data types
MyClient.EnableSegmentData();
MyClient.EnableMarkerData();

% Set the streaming mode
```

```
59 MyClient.SetStreamMode( StreamMode.ClientPull );
60
61 % Set the axis mapping
62 MyClient.SetAxisMapping( Direction.Forward, ...
63                          Direction.Left,    ...
64                          Direction.Up );    % Z-up
65
66 tempcount=0;
67 counter=0;
68 flag1=0;
69 DATACORRECTION=0;
70 data=[1000,1000];
71
72 rb1 = [0,0,0];
73 rb2 = [0,0,0];
74 rb3 = [0,0,0];
75 rb4 = [0,0,0];
76 rb5 = [0,0,0];
77 rb6 = [0,0,0];
78 rb7 = [0,0,0];
79 rb8 = [0,0,0];
80
81 %% GET DATA
82 tglobal=tic;
83 while(matcounter <= matrixsize)
84
85     % 1 - Get a frame
86     while MyClient.GetFrame().Result.Value ~= Result.Success
87     end
88     timeglobal = toc(tglobal);
89
90     % Print the frame number
91     Output_GetFrameNumber = MyClient.GetFrameNumber();
92
```

```
93      marker_1_lost = false;
94      marker_2_lost = false;
95      marker_3_lost = false;
96      marker_4_lost = false;
97      marker_5_lost = false;
98      marker_6_lost = false;
99      marker_7_lost = false;
100     marker_8_lost = false;
101
102     % 2 - Get the subject
103     SubjectCount = MyClient.GetSubjectCount().SubjectCount;
104
105     for SubjectIndex = 1:SubjectCount
106         timer=tic;
107         SubjectName = MyClient.GetSubjectName( SubjectIndex ).
               SubjectName;
108
109         % 3 - get origin data
110         if strcmpi(SubjectName, 'origin')
111             % Get translation data of origin
112             MarkerCount = MyClient.GetMarkerCount( SubjectName ).
                   MarkerCount;
113
114             for MarkerIndex = 1:MarkerCount
115                 MarkerName = MyClient.GetMarkerName( SubjectName,
                        MarkerIndex ).MarkerName;
116
117                 if MarkerName ~= ' ' % error check - it is
                       labelled
118                     MarkerTranslation = MyClient.
                           GetMarkerGlobalTranslation( SubjectName,
                           MarkerName );
119
120                 if strcmpi(MarkerName, 'origin')
```

```
121                         origin = [MarkerTranslation.Translation
                               (1) MarkerTranslation.Translation(2)
                               MarkerTranslation.Translation(3)];
122                     end
123
124                 end
125
126             end
127
128         end % end of get origin data
129
130         % 3 - get robot data
131         if strcmpi(SubjectName, '8MK_3WD_LOW')
132             % Get translation data of all 4 markers
133             MarkerCount = MyClient.GetMarkerCount( SubjectName ).
                    MarkerCount;
134 %             fprintf( '    Markers (%d):\n', MarkerCount );
135             for MarkerIndex = 1:MarkerCount
136                 MarkerName = MyClient.GetMarkerName( SubjectName,
                        MarkerIndex ).MarkerName;
137
138                 if MarkerName ~= ' ' % error check - it is
                        labelled
139
140                     MarkerTranslation = MyClient.
                            GetMarkerGlobalTranslation( SubjectName,
                            MarkerName );
141
142                     if strcmpi(MarkerName, 'rb1')
143
144                         %Added for filling missed frames
145                         if MarkerTranslation.Translation(1) ~= 0
                                && MarkerTranslation.Translation(2) ~=
                                 0 && MarkerTranslation.Translation(3)
```

```
                    ~= 0
146                 rb1 = [MarkerTranslation.Translation
                        (1) MarkerTranslation.Translation
                        (2) MarkerTranslation.Translation
                        (3)];
147                 marker_1_lost = false;
148             else
149                 if matcounter > 3
150                     marker_1_lost = true;
151                 end
152             end
153
154         end
155         if strcmpi(MarkerName, 'rb2')
156
157             %Added for filling missed frames
158             if MarkerTranslation.Translation(1) ~= 0
                    && MarkerTranslation.Translation(2) ~=
                     0 && MarkerTranslation.Translation(3)
                     ~= 0
159                 rb2 = [MarkerTranslation.Translation
                        (1) MarkerTranslation.Translation
                        (2) MarkerTranslation.Translation
                        (3)];
160                 marker_2_lost = false;
161             else
162                 if matcounter > 3
163                     marker_2_lost = true;
164                 end
165             end
166
167         end
168         if strcmpi(MarkerName, 'rb3')
169
```

```
170                         %Added for filling missed frames
171                         if MarkerTranslation.Translation(1) ~= 0
                               && MarkerTranslation.Translation(2) ~=
                                0 && MarkerTranslation.Translation(3)
                                ~= 0
172                             rb3 = [MarkerTranslation.Translation
                                   (1) MarkerTranslation.Translation
                                   (2) MarkerTranslation.Translation
                                   (3)];
173                             marker_3_lost = false;
174                         else
175                             if matcounter > 3
176                                 marker_3_lost = true;
177                             end
178                         end
179
180                     end
181                     if strcmpi(MarkerName, 'rb4')
182
183                         %Added for filling missed frames
184                         if MarkerTranslation.Translation(1) ~= 0
                               && MarkerTranslation.Translation(2) ~=
                                0 && MarkerTranslation.Translation(3)
                                ~= 0
185                             rb4 = [MarkerTranslation.Translation
                                   (1) MarkerTranslation.Translation
                                   (2) MarkerTranslation.Translation
                                   (3)];
186                             marker_4_lost = false;
187                         else
188                             if matcounter > 3
189                                 marker_4_lost = true;
190                             end
191                         end
```

```
192
193                     end
194                     if strcmpi(MarkerName, 'rb5')
195
196                         %Added for filling missed frames
197                         if MarkerTranslation.Translation(1) ~= 0
                               && MarkerTranslation.Translation(2) ~=
                                0 && MarkerTranslation.Translation(3)
                                ~= 0
198                             rb5 = [MarkerTranslation.Translation
                                   (1) MarkerTranslation.Translation
                                   (2) MarkerTranslation.Translation
                                   (3)];
199                             marker_5_lost = false;
200                         else
201                             if matcounter > 3
202                                 marker_5_lost = true;
203                             end
204                         end
205
206                     end
207                     if strcmpi(MarkerName, 'rb6')
208
209                         %Added for filling missed frames
210                         if MarkerTranslation.Translation(1) ~= 0
                               && MarkerTranslation.Translation(2) ~=
                                0 && MarkerTranslation.Translation(3)
                                ~= 0
211                             rb6 = [MarkerTranslation.Translation
                                   (1) MarkerTranslation.Translation
                                   (2) MarkerTranslation.Translation
                                   (3)];
212                             marker_6_lost = false;
213                         else
```

```
214                         if matcounter > 3
215                             marker_6_lost = true;
216                         end
217                     end
218
219                 end
220                 if strcmpi(MarkerName, 'rb7')
221
222                     %Added for filling missed frames
223                     if MarkerTranslation.Translation(1) ~= 0
                           && MarkerTranslation.Translation(2) ~=
                            0 && MarkerTranslation.Translation(3)
                            ~= 0
224                         rb7 = [MarkerTranslation.Translation
                               (1) MarkerTranslation.Translation
                               (2) MarkerTranslation.Translation
                               (3)];
225                         marker_7_lost = false;
226                     else
227                         if matcounter > 3
228                             marker_7_lost = true;
229                         end
230                     end
231
232                 end
233                 if strcmpi(MarkerName, 'rb8')
234
235                     %Added for filling missed frames
236                     if MarkerTranslation.Translation(1) ~= 0
                           && MarkerTranslation.Translation(2) ~=
                            0 && MarkerTranslation.Translation(3)
                            ~= 0
237                         rb8 = [MarkerTranslation.Translation
                               (1) MarkerTranslation.Translation
```

```
                                (2) MarkerTranslation.Translation
                                (3)];
238                         marker_8_lost = false;
239                     else
240                         if matcounter > 3
241                             marker_8_lost = true;
242                         end
243                     end
244
245                 end
246
247             end % end of if for checking to make sure the
                    marker is valid
248
249         end % end of for loop for checking markers
250
251     end % end of robot segment
252
253     % At 100HZ, this value should be fixed at 10ms between
            each
254     % iteration
255     timenow = 0.020;
256
257     %%%-------------set desired position
            ----------------------%%%
258     %fprintf(serialPortObj, '1,0,0*');
259
260     Period=60; %period in seconds
261     rose = 2; %determines number of pedals (2k pedals for
            even k, k pedals for odd k)
262
263     %desired = [1,0,0];
264     %desired = [sin(2*pi*timeglobal/Period),cos(2*pi*
            timeglobal/Period), 0]; %Circle (r=1m)
```

```
265         %desired = [cos(rose*2*pi*timeglobal/Period)*cos(2*pi*
              timeglobal/Period), cos(rose*2*pi*timeglobal/Period)*
              sin(2*pi*timeglobal/Period),0];
266
267         %desired = [0 0 2*pi*timeglobal/12.]; %pspin
268         %desired = [0 0 -2*pi*timeglobal/12.]; %nspin
269         desired = [0 0 sin(2*pi*timeglobal/24.)*0.9*pi]; %sin
270
271         fprintf(serialPortObj, strcat(num2str(desired(1)),",",
              num2str(desired(2)),",",num2str(desired(3)),"*"));
272         desired
273
274         % Save Sheet1 Data
275         format long
276         Sheet1Mat(matcounter,:) = [timeglobal, timenow, rb1, rb2,
               rb3, rb4, rb5, rb6, rb7, rb8, desired]; % Gives raw
              data
277
278         matcounter = matcounter + 1;
279
280     end
281
282 end % end of while loop, everything before this runs until the
      end of the script
283
284
285 desired = [0,0,0];
286 fprintf(serialPortObj, '0,0,0*');
287 %xlswrite("./Raw Data/test_r.xlsx", Sheet1Mat); %Save the data (
      if this isn't working just copy data over to a spread sheet,
      MATLAB is finicky in git repos)
288 xlswrite(notebook_name_raw, Sheet1Mat);
```

Appendix C

Chapter 7 Supplement

This appendix shows all figures of environmental statistics that were calculated, but not included in the body of the thesis, as well as code used for data collection and analysis. All code can be found in the Git repository at https://github.com/dyerbm/OmniWheel.

C.1 Covariances

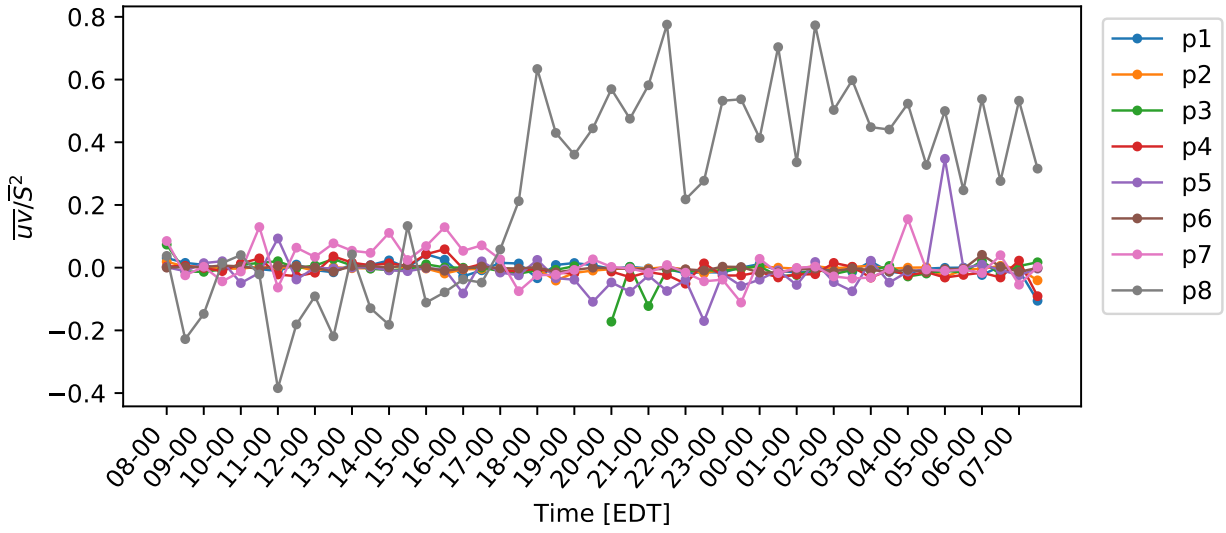

Figure C.1: Normalized covariance between x and y wind velocity components over a diurnal cycle.

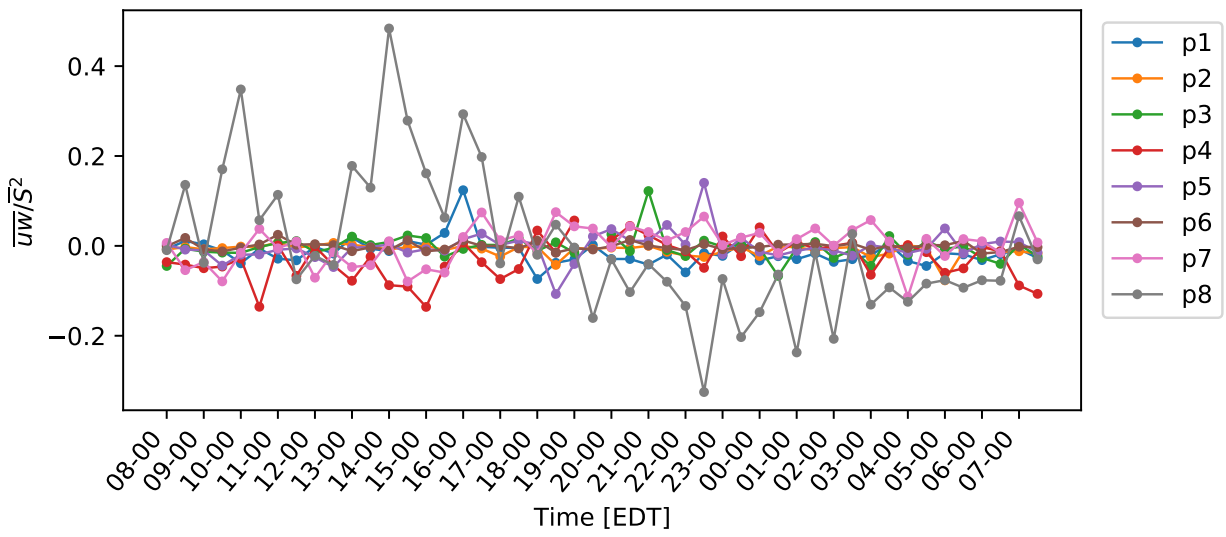

Figure C.2: Normalized covariance between x and z wind velocity components over a diurnal cycle.

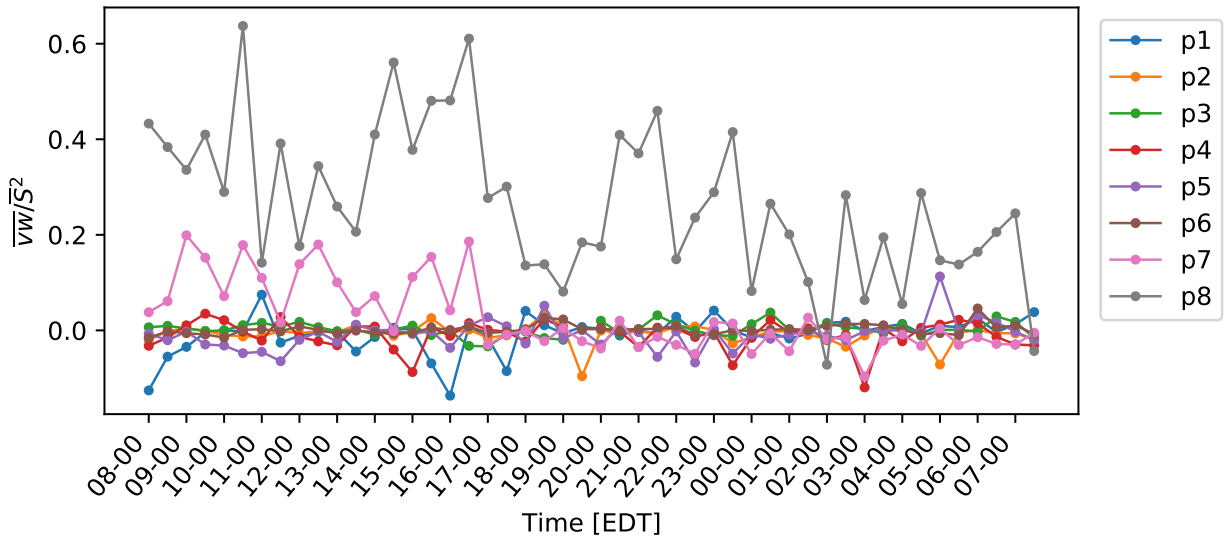

Figure C.3: Normalized covariance between y and z wind velocity components over a diurnal cycle.

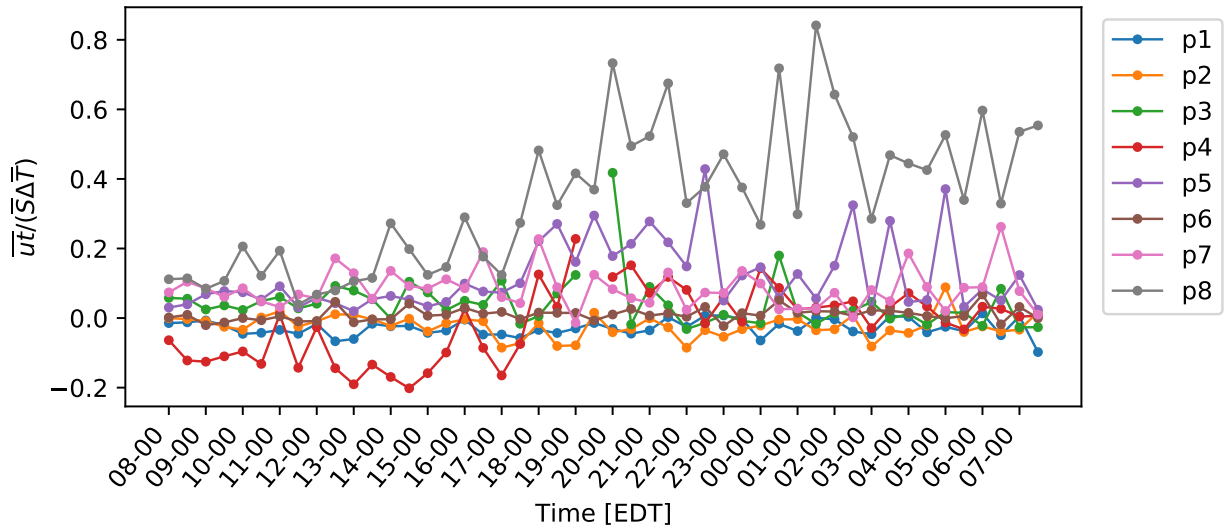

Figure C.4: Normalized covariance between the x wind velocity component and ultrasonic temperature over a diurnal cycle.

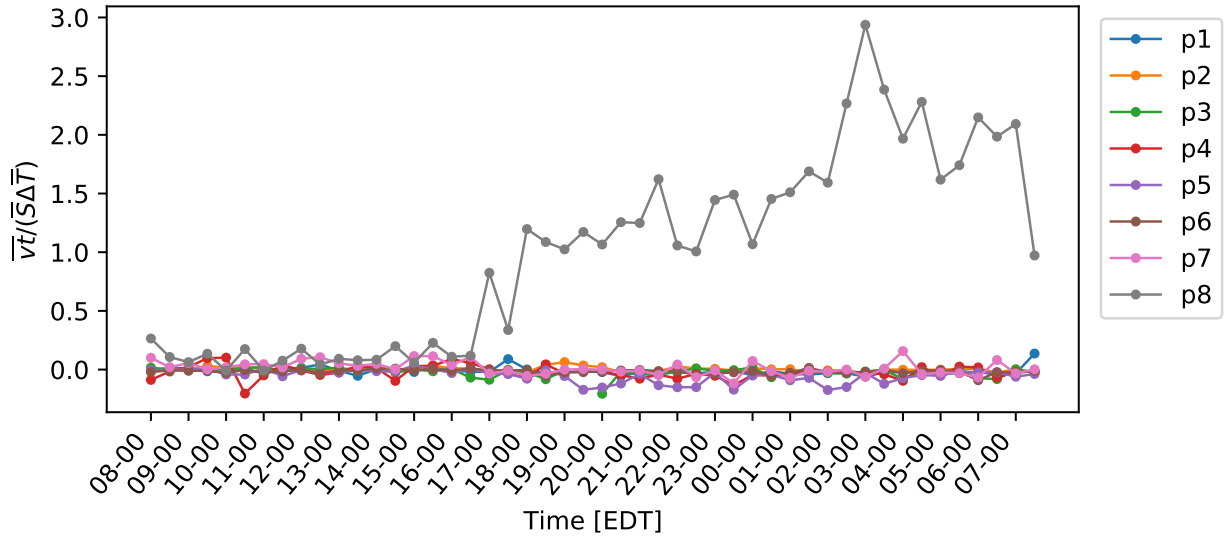

Figure C.5: Normalized covariance between the y wind velocity component and ultrasonic temperature over a diurnal cycle.

C.2 Wiring Diagrams

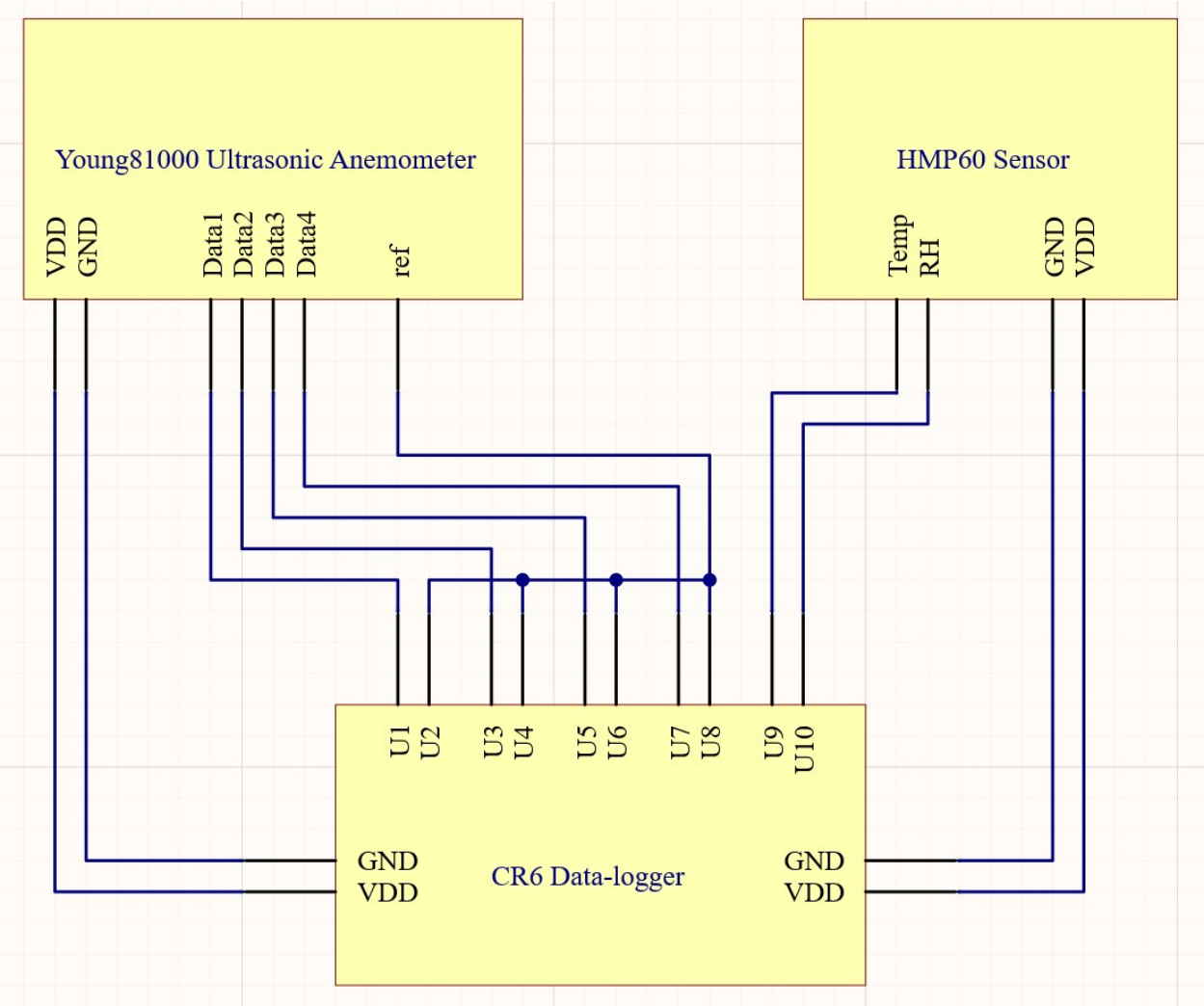

Figure C.6: Wiring diagram of Young81000 ultrasonic anemometer, HMP60 temperature and relative humidity sensor, and CR6 data-logger.

C.3 Codes

C.3.1 Trajectory_Stitch.py

```
import numpy as np
import pandas as pd
```

```
filename = "output.txt"

df = pd.read_csv(filename,
                 delimiter = '\t',
                 names = ['x','y','theta'])

newframe = pd.DataFrame(columns = ['x','y','theta'])

measurement_time=160*50 #time to sample multiplied by controller frequency

dict={}
for pathnum in range(1,9):
    print(pathnum)
    filename = "path"+str(pathnum)+".txt"
    df = pd.read_csv(filename,
                 delimiter = '\t',
                 names = ['x','y','theta'])

    for i in range(len(df)-1): #split into smaller steps using lin spaces
        dictlen=len(dict)
        if (df['x'][i]!=df['x'][i+1] and df['y'][i]!=df['y'][i+1]):
            x=np.linspace(df['x'][i],df['x'][i+1],num=4)
            y=np.linspace(df['y'][i],df['y'][i+1],num=4)
            t=np.linspace(df['theta'][i],df['theta'][i+1],num=4)
            for j in range(3):
                dict[dictlen+j] = {'x':x[j],'y':y[j],'theta':t[j]}
        else:
            x=np.linspace(df['x'][i],df['x'][i+1],num=4)
            y=np.linspace(df['y'][i],df['y'][i+1],num=4)
            t=np.linspace(df['theta'][i],df['theta'][i+1],num=4)
            for j in range(3):
                dict[dictlen+j] = {'x':x[j],'y':y[j],'theta':t[j]}

    dictlen=len(dict)
    for i in range(measurement_time): #fill in time at point
        dict[dictlen+i] = {'x':df['x'].iloc[-1],'y':df['y'].iloc[-1],'theta':
            df['theta'].iloc[-1]}

newframe = pd.DataFrame.from_dict(dict, "index")
```

```
newframe.to_csv('finalpath_slow.csv',index=False)
```

C.3.2 Position_CR6_Joiner.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from datetime import timedelta

position_path = "./Data/"
data_path = "./Data/RealRun/"
output_path ="./"



'''Create single CR6 dataframe with datetime for comparisons'''
dict={}
for i in tqdm(range(5462,13774)): #create dataframe of all CR6 data
    try:
        outputsize = len(dict)

        df_CR6 = pd.read_table(data_path+"Younng10Hz"+str(i)+".dat",
                               sep=",",
                              skiprows=[0,2,3])

        for j in range(len(df_CR6)):
            if "." in df_CR6['TIMESTAMP'][j]:
                date = datetime.strptime(df_CR6['TIMESTAMP'][j], '%Y-%m-%d %H
                    :%M:%S.%f')
            else:
                date = datetime.strptime(df_CR6['TIMESTAMP'][j], '%Y-%m-%d %H
                    :%M:%S')

            dict[outputsize+j]={"TIMESTAMP":date,"Record":df_CR6['RECORD'][j],
                "U (m s^-1)":df_CR6['U'][j],"V (m s^-1)":df_CR6['V'][j],"W (m
                s^-1)":df_CR6['W'][j],"TSonic (K)":df_CR6['TSonic'][j],"RH (%)
                ":df_CR6['RH'][j],"T_HMP60 (C)":df_CR6['T_HMP60'][j]}
```

```
    except: #if the file is missing just skip it (this happens twice, nothing
        needed luckily)
         pass

df_CR6 = pd.DataFrame.from_dict(dict, "index") #convert dict to dataframe (
    this is faster)

'''START LOOP HERE!!!'''
inputFiles = ['2020-10-02 16-00.xlsx','2020-10-02 16-30.xlsx','2020-10-02
    17-00.xlsx','2020-10-02 17-31.xlsx','2020-10-02 18-00.xlsx','2020-10-02
    18-30.xlsx','2020-10-02 19-00.xlsx','2020-10-02 19-30.xlsx','2020-10-02
    20-00.xlsx','2020-10-02 20-30.xlsx','2020-10-02 21-00.xlsx','2020-10-02
    21-30.xlsx','2020-10-02 22-00.xlsx','2020-10-02 22-30.xlsx','2020-10-02
    23-00.xlsx','2020-10-02 23-30.xlsx','2020-10-03 00-00.xlsx','2020-10-03
    00-30.xlsx','2020-10-03 01-00.xlsx','2020-10-03 01-30.xlsx','2020-10-03
    02-00.xlsx','2020-10-03 02-30.xlsx','2020-10-03 03-00.xlsx','2020-10-03
    03-30.xlsx','2020-10-03 04-00.xlsx','2020-10-03 04-30.xlsx','2020-10-03
    05-00.xlsx','2020-10-03 05-30.xlsx','2020-10-03 06-00.xlsx','2020-10-03
    06-30.xlsx', '2020-10-03 07-00.xlsx','2020-10-03 07-30.xlsx','2020-10-08
    08-00.xlsx','2020-10-08 08-30.xlsx','2020-10-08 09-00.xlsx','2020-10-08
    09-30.xlsx','2020-10-08 10-00.xlsx','2020-10-08 10-30.xlsx','2020-10-08
    11-00.xlsx','2020-10-08 11-30.xlsx','2020-10-08 12-00.xlsx','2020-10-08
    12-30.xlsx','2020-10-08 13-00.xlsx','2020-10-08 13-30.xlsx','2020-10-08
    14-00.xlsx','2020-10-08 14-30.xlsx','2020-10-08 15-00.xlsx','2020-10-02
    15-30.xlsx']


outputFiles = ['16-00','16-30','17-00','17-30','18-00','18-30','19-00','19-30'
    ,'20-00','20-30','21-00','21-30','22-00','22-30','23-00','23-30','00-00','
    00-30','01-00','01-30','02-00','02-30','03-00','03-30','04-00','04-30','
    05-00','05-30','06-00','06-30','07-00','07-30','08-00','08-30','09-00','
    09-30','10-00','10-30','11-00','11-30','12-00','12-30','13-00','13-30','
    14-00','14-30','15-00','15-30']


for j in tqdm(range(len(inputFiles))):
    df_pos = pd.read_excel(position_path+inputFiles[j], header=None)
    df_pos.columns = ['TIMESTAMP', 'X', 'Y', 'THETA']

    '''Create proper timestamps for position'''
    dict ={}
```

```
for i in tqdm(range(len(df_pos))):
    dict[i]={'TIMESTAMP': datetime.fromordinal(int(df_pos['TIMESTAMP'][i])
        ) + timedelta(days=df_pos['TIMESTAMP'][i]%1) - timedelta(days =
        366),'X':df_pos['X'][i],'Y':df_pos['Y'][i],'THETA':df_pos['THETA'
        ][i]}
df_pos = pd.DataFrame.from_dict(dict, "index")



'''Trim CR6 data based on timestamps from position data. Use new dataframe
    .'''

df_CR6_trim= df_CR6.iloc[df_CR6.index[(df_CR6['TIMESTAMP'] == df_pos['
    TIMESTAMP'].iloc[0].round('100ms'))].tolist()[0]:
                        df_CR6.index[(df_CR6['TIMESTAMP'] == df_pos['
                            TIMESTAMP'].iloc[-1].round('100ms'))].tolist()
                            [0]]

df_CR6_trim.reset_index(inplace=True, drop=True)


'''Create a combined database'''
dict={}
for i in tqdm(range(len(df_CR6_trim))):
    date = df_CR6_trim["TIMESTAMP"].iloc[i]
    index = (df_pos['TIMESTAMP']-date).abs().argsort()[:2].iloc[1]

    x = df_pos['X'].iloc[index]
    y = df_pos['Y'].iloc[index]
    theta = df_pos['THETA'].iloc[index]

    dict[i]= {"Year":date.year,"Month":date.month,"Day":date.day,"Hour":
        date.hour,
                "Minute":date.minute,"Second":date.second+date.microsecond
                    /1000000.,
                "x":x,"y":y,"theta":theta,
                "Record":df_CR6_trim['Record'][i],"U (m s^-1)":df_CR6_trim['
                    U (m s^-1)'][i],"V (m s^-1)":df_CR6_trim['V (m s^-1)'][i
                    ],
                "W (m s^-1)":df_CR6_trim['W (m s^-1)'][i],"TSonic (K)":
                    df_CR6_trim['TSonic (K)'][i],"RH (%)":df_CR6_trim['RH
```

```
                (%)'][i],
              "T_HMP60_(C)":df_CR6_trim['T_HMP60_(C)'][i]}


df_comb = pd.DataFrame.from_dict(dict, "index")


'''Create MultiIndex based on each point'''
point_dict={'p1':[2.8,0.7,0], 'p2':[6.0,0.7,0], 'p3':[6.0,4.4,0], 'p4'
    :[2.8,4.4,0],
          'p5':[6.0,7.9,0], 'p6':[2.8,7.9,0], 'p7':[6.0,10.6,0], 'p8'
             :[4.9,11.8,0]}

drop=50 #frequency*seconds to drop, make sure the robot is in the correct
    positon

dict={'p1':{},'p2':{},'p3':{},'p4':{},'p5':{},'p6':{},'p7':{},'p8':{}}

for i in tqdm(range(drop, len(df_comb)-drop)):
    if ([df_comb['x'].iloc[i-drop],df_comb['y'].iloc[i-drop],df_comb['
        theta'].iloc[i-drop]] in point_dict.values() and
       [df_comb['x'].iloc[i+drop],df_comb['y'].iloc[i+drop],df_comb['theta
           '].iloc[i+drop]] in point_dict.values() and
       [df_comb['x'].iloc[i],df_comb['y'].iloc[i],df_comb['theta'].iloc[i
           ]] in point_dict.values()): #Check that it has been there for a
            while

        point = list(point_dict.keys())[list(point_dict.values()).index([
            df_comb['x'].iloc[i],df_comb['y'].iloc[i],df_comb['theta'].
            iloc[i]])] #get the point

        dict[point][i]=df_comb.iloc[i].to_dict()


dict_of_df = {k: pd.DataFrame(v) for k,v in dict.items()}
output = pd.concat(dict_of_df, axis=1).transpose()
output.to_csv('./Data/Binned_Data/'+outputFiles[j]+'.csv',index=True) #
    save dat data
```

C.3.3 YOUNG81000_HMP60_10Hz.CR6

```
'To create a different opening program template, type in new
'instructions and select Template | Save as Default Template

'Date: 2020-09-20
'Program author: Benjamin Dyer


PipeLineMode

Public PTemp_C, BattV, DiffVolt_1, DiffVolt_2, DiffVolt_3, DiffVolt_4, T_HMP60
    , RH
Dim Flag As Boolean

Alias DiffVolt_1 = U
Alias DiffVolt_2 = V
Alias DiffVolt_3 = W
Alias DiffVolt_4 = TSonic

Units BattV=Volts
Units PTemp_C=Deg C
Units DiffVolt_1=mV
Units DiffVolt_2=mV
Units DiffVolt_3=mV
Units DiffVolt_4=mV
Units T_HMP60=Deg C
Units RH=%

'Declare Private Variables
'Example:
'Dim Counter

'Define Data Tables
DataTable (Young10Hz_hmp,True,10000) 'Set table size to # of records, or -1 to
     autoallocate.
  TableFile ("CRD:Younng10Hz",8,-1,600,0,Min,0,0)
        DataInterval (0,100,mSec,10)
  'TableFile ("CRD:Test",64,-1,0,60,Min,0,0)
        Sample(1,DiffVolt_1,FP2)
        Sample(1,DiffVolt_2,FP2)
        Sample(1,DiffVolt_3,FP2)
        Sample(1,DiffVolt_4,FP2)
```

```
        Sample(1,RH,FP2)
        Sample(1,T_HMP60,FP2)
EndTable

DataTable(CR6_tab,True,-1)
  DataInterval(0,1,Min,10)
  Minimum(1,BattV,FP2,False,False)
EndTable

'Define Subroutines
'Sub
        'EnterSub instructions here
'EndSub

'Main Program
BeginProg
        Scan (100,mSec,8,0)
                VoltDiff(DiffVolt_1,1,mV5000,U1,False,20,15000,0.004,-10)
                VoltDiff(DiffVolt_2,1,mV5000,U3,False,20,15000,0.004,-10)
                VoltDiff(DiffVolt_3,1,mV5000,U5,False,20,15000,0.004,-10)
                VoltDiff(DiffVolt_4,1,mV5000,U7,False,20,15000,0.02,220)

    VoltSe(T_HMP60,1,mV1000,U9,False,0,60,0.1,-40)
                VoltSe(RH,1,mV1000,U10,False,0,60,0.1,0)
                If (RH>100) AND (RH<108) Then RH=100
    If TimeIntoInterval(0,5,Sec) Then Flag=true

    CallTable Young10Hz_hmp
    CallTable CR6_tab

        NextScan

EndProg
```

C.3.4 Analysis.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
```

```
from datetime import timedelta
import matplotlib.ticker as ticker
import itertools
from numpy.lib import scimath as SM

'''Create empty Dataframe'''
my_index = pd.MultiIndex(levels=[[],[],[]],
                         codes=[[],[],[]],
                             names=[u'STime', u'Point', u'Index'])
my_columns = [u'Year', u'Month',u'Day',u'Hour',u'Minute',u'Second',u'x',u'y',u
    'theta',u'Record',u'U (m s^-1)',u'V (m s^-1)',u'W (m s^-1)',u'TSonic (K)',
    u'RH (%)',u'T_HMP60 (C)']
data = pd.DataFrame(index=my_index, columns=my_columns)

'''IMPORT DATA'''
files = ['08-00','08-30','09-00','09-30','10-00','10-30','11-00','11-30',
         '12-00','12-30','13-00','13-30','14-00','14-30','15-00','15-30',
         '16-00','16-30','17-00','17-30','18-00','18-30','19-00','19-30',
         '20-00','20-30','21-00','21-30','22-00','22-30','23-00','23-30',
         '00-00','00-30','01-00','01-30','02-00','02-30','03-00','03-30',
         '04-00','04-30','05-00','05-30','06-00','06-30','07-00','07-30']

for i in tqdm(files):
    temppd = pd.read_csv('./Data/Binned Data/'+i+'.csv',index_col=[0,1])

    #append dataframe into multiframe
    data = data.append(temppd.assign(k=i).set_index('k',append=True).swaplevel
        (0,2).swaplevel(1,2))

def crosscorr(x,y,lag):

    xmean = np.mean(x)
    ymean = np.mean(y)
    N = len(x)
    cross = np.zeros((N,1))
    tot = 0

    for j in range(lag):
        temp1=0
        temp2=0
        temp3=0
```

```
            xk=0
            yk=0
            xkj=0
            ykj=0
            for k in range(N-j-1):
                xk+=1/(N-j)*x[k]
                yk+=1/(N-j)*y[k]
                xkj+=1/(N-j)*x[k+j]
                ykj+=1/(N-j)*y[k+j]
            for k in range(N-j-1):
                temp1 += (x[k]-xk)*(y[k+j]-ykj)
                temp2 += (x[k]-xk)*(y[k]-yk)
                temp3 += (x[k+j]-xkj)*(y[k+j]-ykj)

            cross[j]=temp1/(SM.sqrt(temp2)*SM.sqrt(temp3))

            if np.sign(cross[0])!=np.sign(cross[j]):
                return sum(cross)

        return lag

    def psum(x,sign):
        sum = 0
        sign = np.sign(x[0])
        if sign==-1:
            x = [ -i for i in x ]
        for i in range(len(x)):
            if x[i]<0:
                return sum*sign
            else:
                sum+=x[i]

        return sum*sign


    points=['p1','p2','p3','p4','p5','p6','p7','p8']

    my_index = pd.MultiIndex(levels=[[],[]],
                             codes=[[],[]],
                                 names=[u'STime', u'Point'])
    my_columns = [u'Uavg',u'Vavg',u'Wavg','Savg',u'TSonicavg',u'RHavg',u'
```

```
    T_HMP60avg','u'Uvar','u'Uvar_rerr','u'Vvar','u'Vvar_rerr','u'Wvar','u'Wvar_rerr'
    ,u'TSonicvar',u'TSonicvar_rerr',
                u'k',u'UVvar',u'UVvar_rerr',u'UWvar',u'UWvar_rerr',u'VWvar',u'
                    VWvar_rerr',
                u'UTvar',u'UTvar_rerr',u'VTvar',u'VTvar_rerr',u'WTvar',u'
                    WTvar_rerr',
               u'PMV1',u'PPD1',u'PMV2',u'PPD2',u'PMV3',u'PPD3',u'PMV4',u'PPD4',u
                   'PMV5',u'PPD5',u'PMV6',u'PPD6',]

p_data = pd.DataFrame(index=my_index, columns=my_columns)

NIntegral = 600
AverageSample = len(data.loc[('08-00','p1'),'U_(m_s^-1)'])
WindowLength=int(AverageSample/2)
dt=0.1
SampleTime=AverageSample/10.

RUU=np.zeros((NIntegral,1))
RVV=np.zeros((NIntegral,1))
RWW=np.zeros((NIntegral,1))
RTT=np.zeros((NIntegral,1))
RUV=np.zeros((NIntegral,1))
RVW=np.zeros((NIntegral,1))
RUW=np.zeros((NIntegral,1))
RUT=np.zeros((NIntegral,1))
RVT=np.zeros((NIntegral,1))
RWT=np.zeros((NIntegral,1))

RUUALL=np.zeros((WindowLength,1))
RVVALL=np.zeros((WindowLength,1))
RWWALL=np.zeros((WindowLength,1))
RTTALL=np.zeros((WindowLength,1))
RUVALL=np.zeros((WindowLength,1))
RVWALL=np.zeros((WindowLength,1))
RUWALL=np.zeros((WindowLength,1))
RUTALL=np.zeros((WindowLength,1))
RVTALL=np.zeros((WindowLength,1))
RWTALL=np.zeros((WindowLength,1))


for dtime in tqdm(files):
```

```
for point in points:
    p_data.loc[(dtime,point),'Uavg'] = np.mean(data.loc[(dtime,point),'U (
        m s^-1)'])
    p_data.loc[(dtime,point),'Vavg'] = np.mean(data.loc[(dtime,point),'V (
        m s^-1)'])
    p_data.loc[(dtime,point),'Wavg'] = np.mean(data.loc[(dtime,point),'W (
        m s^-1)'])
    p_data.loc[(dtime,point),'TSonicavg'] = np.mean(data.loc[(dtime,point)
        ,'TSonic (K)'])
    p_data.loc[(dtime,point),'RHavg'] = np.mean(data.loc[(dtime,point),'RH
        (%)'])
    p_data.loc[(dtime,point),'T_HMP60avg'] = np.mean(data.loc[(dtime,point
        ),'T_HMP60 (C)'])+273.15 #shift to Kelvin
    p_data.loc[(dtime,point),'Savg'] = np.mean(np.sqrt(data.loc[(dtime,
        point),'U (m s^-1)']**2+
                                                       data.loc[(dtime,
                                                           point),'V (m s
                                                           ^-1)']**2+
                                                       data.loc[(dtime,
                                                           point),'W (m s
                                                           ^-1)']**2))


    x = [j for j in range(0, len(data.loc[(dtime,point),'U (m s^-1)']))]
    U = data.loc[(dtime,point),'U (m s^-1)']
    Umodel = np.polyfit(x,U,1)
    Utrend = np.polyval(Umodel,x)
    Udetrended = U - Utrend
    V = data.loc[(dtime,point),'V (m s^-1)']
    Vmodel = np.polyfit(x,V,1)
    Vtrend = np.polyval(Vmodel,x)
    Vdetrended = V - Vtrend
    W = data.loc[(dtime,point),'W (m s^-1)']
    Wmodel = np.polyfit(x,W,1)
    Wtrend = np.polyval(Wmodel,x)
    Wdetrended = W - Wtrend
    TSonic =  data.loc[(dtime,point),'TSonic (K)']
    TSonicmodel = np.polyfit(x, TSonic,1)
    TSonictrend = np.polyval( TSonicmodel,x)
    TSonicdetrended =  TSonic -  TSonictrend
```

```
UVCovMatrix = np.cov(Udetrended, Vdetrended)
UWCovMatrix = np.cov(Udetrended, Wdetrended)
VWCovMatrix = np.cov(Vdetrended, Wdetrended)
UTSonicCovMatrix = np.cov(Udetrended, TSonicdetrended)
VTSonicCovMatrix = np.cov(Vdetrended, TSonicdetrended)
WTSonicCovMatrix = np.cov(Wdetrended, TSonicdetrended)

p_data.loc[(dtime,point),'Uvar'] = UVCovMatrix[0,0]
p_data.loc[(dtime,point),'Vvar'] = UVCovMatrix[1,1]
p_data.loc[(dtime,point),'Wvar'] = UWCovMatrix[1,1]
p_data.loc[(dtime,point),'TSonicvar'] = UTSonicCovMatrix[1,1]
p_data.loc[(dtime,point),'k'] = 1/2*(p_data.loc[(dtime,point),'Uvar']+
    p_data.loc[(dtime,point),'Vvar']+p_data.loc[(dtime,point),'Wvar'])

p_data.loc[(dtime,point),'UVvar'] = UVCovMatrix[0,1]
p_data.loc[(dtime,point),'UWvar'] = UWCovMatrix[0,1]
p_data.loc[(dtime,point),'VWvar'] = VWCovMatrix[0,1]
p_data.loc[(dtime,point),'UTvar'] = UTSonicCovMatrix[0,1]
p_data.loc[(dtime,point),'VTvar'] = VTSonicCovMatrix[0,1]
p_data.loc[(dtime,point),'WTvar'] = WTSonicCovMatrix[0,1]

Udetrended = Udetrended.to_numpy()
Vdetrended = Vdetrended.to_numpy()
Wdetrended = Wdetrended.to_numpy()
TSonicdetrended = TSonicdetrended.to_numpy()


#Calculate integral scales (these are typically on the scale of 1-5
    seconds)
TauUU = crosscorr(Udetrended,Udetrended, NIntegral)*dt
TauVV = crosscorr(Vdetrended,Vdetrended, NIntegral)*dt
TauWW = crosscorr(Wdetrended,Wdetrended, NIntegral)*dt
TauTT = crosscorr(TSonicdetrended,TSonicdetrended, NIntegral)*dt
TauUV = crosscorr(Udetrended,Vdetrended, NIntegral)*dt
TauUW = crosscorr(Udetrended,Wdetrended, NIntegral)*dt
TauVW = crosscorr(Vdetrended,Wdetrended, NIntegral)*dt
TauUT = crosscorr(Udetrended,TSonicdetrended, NIntegral)*dt
TauVT = crosscorr(Vdetrended,TSonicdetrended, NIntegral)*dt
TauWT = crosscorr(Wdetrended,TSonicdetrended, NIntegral)*dt

#Adjust for Systematic Error
```

```
            p_data.loc[(dtime,point),'Uvar'] *=2*TauUU/SampleTime+1
            p_data.loc[(dtime,point),'Vvar'] *=2*TauVV/SampleTime+1
            p_data.loc[(dtime,point),'Wvar'] *=2*TauWW/SampleTime+1
            p_data.loc[(dtime,point),'TSonicvar'] *=2*TauTT/SampleTime+1
            p_data.loc[(dtime,point),'UVvar'] *=2*TauUV/SampleTime+1
            p_data.loc[(dtime,point),'UWvar'] *=2*TauUW/SampleTime+1
            p_data.loc[(dtime,point),'VWvar'] *=2*TauVW/SampleTime+1
            p_data.loc[(dtime,point),'UTvar'] *=2*TauUT/SampleTime+1
            p_data.loc[(dtime,point),'VTvar'] *=2*TauVT/SampleTime+1
            p_data.loc[(dtime,point),'WTvar'] *=2*TauWT/SampleTime+1

            #Save the Random Error (In case you want to check it)
            p_data.loc[(dtime,point),'Uvar_rerr'] =np.sqrt(2*TauUU/SampleTime)
            p_data.loc[(dtime,point),'Vvar_rerr'] =np.sqrt(2*TauVV/SampleTime)
            p_data.loc[(dtime,point),'Wvar_rerr'] =np.sqrt(2*TauWW/SampleTime)
            p_data.loc[(dtime,point),'TSonicvar_rerr'] =np.sqrt(2*TauTT/SampleTime
                )
            p_data.loc[(dtime,point),'UVvar_rerr'] =np.sqrt(2*TauUV/SampleTime)
            p_data.loc[(dtime,point),'UWvar_rerr'] =np.sqrt(2*TauUW/SampleTime)
            p_data.loc[(dtime,point),'VWvar_rerr'] =np.sqrt(2*TauVW/SampleTime)
            p_data.loc[(dtime,point),'UTvar_rerr'] =np.sqrt(2*TauUT/SampleTime)
            p_data.loc[(dtime,point),'VTvar_rerr'] =np.sqrt(2*TauVT/SampleTime)
            p_data.loc[(dtime,point),'WTvar_rerr'] =np.sqrt(2*TauWT/SampleTime)


'''Calibrate the USonic temperatures'''

#plt.scatter(p_data['TSonicavg'],p_data['T_HMP60avg']) #check pre-calibration
    if desired
#plt.show()

#print(p_data['TSonicavg'].tolist())
results = {}
x=p_data['TSonicavg'].tolist()
y=p_data['T_HMP60avg'].tolist()

coeffs = np.polyfit(x,y, 1)

results['polynomial'] = coeffs.tolist()
# r-squared
p = np.poly1d(coeffs)
```

```
# fit values, and mean
yhat = p(x)                         # or [p(z) for z in x]
ybar = np.sum(y)/len(y)          # or sum(y)/len(y)
ssreg = np.sum((yhat-ybar)**2)   # or sum([ (yihat - ybar)**2 for yihat in
    yhat])
sstot = np.sum((y - ybar)**2)    # or sum([ (yi - ybar)**2 for yi in y])
results['determination'] = ssreg / sstot

print(results)

p_data['TSonicavg']=p_data['TSonicavg']*results['polynomial'][0]+results['
    polynomial'][1]


def calculatePmv(ta, tr, vel, rh, met, clo, wme): #THIS CALCULATES PMV AND PPD
    (SUPER USEFUL!)
      #returns [pmv, ppd]
      #ta, air temperature (C)
      #tr, mean radiant temperature (C)
      #vel, relative air speed (m/s)
      #rh, relative humidity (%) Used only this way to input humidity level
      #met, metabolic rate (met)
      #clo, clothing (clo)
      #wme, external work, normally around 0 (met)

    pa = rh * 10 * np.exp(16.6536 - 4030.183 / (ta + 235))

    icl = 0.155 * clo #thermal insulation of the clothing in [m^2 K W^-1]
    m = met * 58.15 #metabolic rate in [W m^-2]
    w = wme * 58.15 #external work in [W m^-2]
    mw = m - w #internal heat production in the human body

    fcl = 1 + 1.29 * icl if icl <= 0.078 else 1.05 + 0.645 * icl

    #heat transf. coeff. by forced convection
    hcf = 12.1 * np.sqrt(vel)
    taa = ta + 273
    tra = tr + 273
    #we have verified that using the equation below or this tcla = taa + (35.5
        - ta) / (3.5 * (6.45 * icl + .1)) does not affect the PMV value
    tcla = taa + (35.5 - ta) / (3.5 * icl + 0.1)
```

```
p1 = icl * fcl
p2 = p1 * 3.96
p3 = p1 * 100
p4 = p1 * taa
p5 = 308.7 - 0.028 * mw + p2 * (tra / 100) ** 4

xn = tcla / 100
xf = tcla / 50
eps = 0.00015

n = 0
hc = 0
while (np.abs(xn - xf) > eps):
    xf = (xf + xn) / 2
    hcn = 2.38 * np.abs(100.0 * xf - taa) ** 0.25
    hc = hcf if hcf > hcn else hcn

    xn = (p5 + p4 * hc - p2 * xf ** 4) / (100 + p3 * hc)
    n+=1
    if (n > 150):
        print("Max iterations exceeded")
        return 1



tcl = 100 * xn - 273;

#heat loss diff. through skin
hl1 = 3.05 * 0.001 * (5733 - 6.99 * mw - pa)
#heat loss by sweating
hl2 = 0.42 * (mw - 58.15) if (mw > 58.15) else 0
#latent respiration heat loss
hl3 = 1.7 * 0.00001 * m * (5867 - pa)
#dry respiration heat loss
hl4 = 0.0014 * m * (34 - ta)
#heat loss by radiation
hl5 = 3.96 * fcl * (xn ** 4 - (tra / 100) ** 4)
#heat loss by convection
hl6 = fcl * hc * (tcl - ta)
```

```
    ts = 0.303 * np.exp(-0.036 * m) + 0.028
    pmv = ts * (mw - hl1 - hl2 - hl3 - hl4 - hl5 - hl6)
    ppd = 100.0 - 95.0 * np.exp(-0.03353 * pmv ** 4.0 - 0.2179 * pmv ** 2.0)

    return [pmv,ppd]




for dtime in tqdm(files):
    for point in points:
        PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
            loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
            Savg'], p_data.loc[(dtime,point),'RHavg'], 1.1, 0.57, 0)
        p_data.loc[(dtime,point),'PMV1'] = PMV[0]
        p_data.loc[(dtime,point),'PPD1'] = PMV[1]

        PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
            loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
            Savg'], p_data.loc[(dtime,point),'RHavg'], 1.1, 0.74, 0)
        p_data.loc[(dtime,point),'PMV2'] = PMV[0]
        p_data.loc[(dtime,point),'PPD2'] = PMV[1]

        PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
            loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
            Savg'], p_data.loc[(dtime,point),'RHavg'], 1.1, 0.96, 0)
        p_data.loc[(dtime,point),'PMV3'] = PMV[0]
        p_data.loc[(dtime,point),'PPD3'] = PMV[1]

        PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
            loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
            Savg'], p_data.loc[(dtime,point),'RHavg'], 1.7, 0.57, 0)
        p_data.loc[(dtime,point),'PMV4'] = PMV[0]
        p_data.loc[(dtime,point),'PPD4'] = PMV[1]

        PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
            loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
            Savg'], p_data.loc[(dtime,point),'RHavg'], 1.7, 0.74, 0)
        p_data.loc[(dtime,point),'PMV5'] = PMV[0]
```

```
            p_data.loc[(dtime,point),'PPD5'] = PMV[1]

            PMV= calculatePmv(p_data.loc[(dtime,point),'TSonicavg']-273, p_data.
                loc[(dtime,point),'TSonicavg']-273+0.3, p_data.loc[(dtime,point),'
                Savg'], p_data.loc[(dtime,point),'RHavg'], 1.7, 0.96, 0)
            p_data.loc[(dtime,point),'PMV6'] = PMV[0]
            p_data.loc[(dtime,point),'PPD6'] = PMV[1]

'''Below is for saving figures'''
figsavepath = './Analysis/Figures/24Hr_'


x_axis_labels = ['16-00','17-00','18-00','19-00',
                 '20-00','21-00','22-00','23-00',
                 '00-00','01-00','02-00','03-00',
                 '04-00','05-00','06-00','07-00',
                 '08-00','09-00','10-00','11-00',
                 '12-00','13-00','14-00','15-00']

marker = itertools.cycle(('.'))
colors = {'p1':'b'}



fig = plt.figure(figsize=(7,3),dpi=600)
ax=plt.subplot(111)
for i in points:
    ax.plot(files,p_data.loc[pd.IndexSlice[:, i], 'T_HMP60avg'],marker = next(
        marker),linewidth=1,label=i)
plt.ylabel(r'$\overline{T}_{HMP60}$ [K]')
xlabel = plt.xlabel('Time [EDT]')
plt.xticks(rotation=40,ha="right")
lgd = ax.legend(bbox_to_anchor=(1.01, 1), loc='upper left')
plt.xticks(x_axis_labels)
plt.savefig(figsavepath+'HMP60_T_avg.pdf', format='pdf',bbox_extra_artists=(
    lgd,xlabel), bbox_inches='tight')
plt.show()


fig = plt.figure(figsize=(7,3),dpi=300)
ax=plt.subplot(111)
```

```
for i in points:
    ax.plot(files,p_data.loc[pd.IndexSlice[:, i], 'TSonicavg'],marker = next(
        marker),linewidth=1,label=i)
plt.ylabel(r'$\overline{T}_{US}$ [K]')
xlabel = plt.xlabel('Time [EDT]')
plt.xticks(rotation=50,ha="right")
lgd = ax.legend(bbox_to_anchor=(1.01, 1), loc='upper left')
plt.xticks(x_axis_labels)
plt.savefig(figsavepath+'SonicT_avg.pdf', format='pdf',bbox_extra_artists=(lgd
    ,xlabel), bbox_inches='tight')
plt.show()


fig = plt.figure(figsize=(7,3),dpi=300)
ax=plt.subplot(111)
for i in points:
    ax.plot(files,p_data.loc[pd.IndexSlice[:, i], 'RHavg'],marker = next(
        marker),linewidth=1,label=i)
plt.ylabel(r'$\overline{RH}$ [%]')
xlabel = plt.xlabel('Time [EDT]')
plt.xticks(rotation=60,ha="right")
lgd = ax.legend(bbox_to_anchor=(1.01, 1), loc='upper left')
plt.xticks(x_axis_labels)
plt.savefig(figsavepath+'RH_avg.pdf', format='pdf',bbox_extra_artists=(lgd,
    xlabel), bbox_inches='tight')
plt.show()
```