5-1-2018

Machine-Insect Interface: Spatial Navigation of a Mobile Robot by a Drosophila

Suddarsun Shivakumar
Southern Illinois University Carbondale, suddarsun@gmail.com

MACHINE-INSECT INTERFACE: SPATIAL NAVIGATION OF A MOBILE ROBOT BY A *DROSOPHILA*

by

Suddarsun Shivakumar

B.S., Southern Illinois University, 2016

A Thesis
Submitted in Partial Fulfillment of the Requirements for the
Master of Science Degree in Mechanical Engineering

Department of Mechanical Engineering and Energy Processes
in the Graduate School
Southern Illinois University Carbondale
May 2018

THESIS APPROVAL

MACHINE-INSECT INTERFACE: SPATIAL NAVIGATION OF A MOBILE ROBOT BY A *DROSOPHILA*

By

Suddarsun Shivakumar

A Thesis Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Master of Science

in the field of Mechanical Engineering

Approved by

Dr. Dal Hyung Kim, Chair

Dr. Farhan Chowdhury

Dr. James Mathias

Graduate School
Southern Illinois University Carbondale
April 5, 2018

AN ABSTRACT OF THE THESIS OF

Suddarsun Shivakumar, for the Master of Science degree in Mechanical Engineering, presented on April 5, 2018, at Southern Illinois University Carbondale.

TITLE: MACHINE-INSECT INTERFACE: SPATIAL NAVIGATION OF A MOBILE ROBOT BY A *DROSOPHILA*

MAJOR PROFESSOR: Dr. Dal Hyung Kim

Machine-insect interfaces have been studied in detail in the past few decades. Animal-machine interfaces have been developed in various ways. In our study, we develop a machine-insect interface wherein an untethered fruit fly (*Drosophila melanogaster*) is tracked to remotely control a mobile robot. We develop the Active Omni-directional Treadmill (AOT) model, and integrate into the mobile robot to create the interface between the robot and the fruit fly. In this system, a fruit fly is allowed to walk on top of a transparent ball. As the fly tries to walk on the ball, we track the position of the fly using the dark field imaging technique. The displacement of the fly will be balanced out by a counter-displacement of the transparent ball, which is actuated by the omni-directional wheels, to keep the fly at the same position on the ball. Then the mobile robot spatially navigates based on the fly movements. The Robotic Operating System (ROS) is used to interface between the ball tracker and the mobile robot wirelessly. This study will help in investigating the fly's behavior under different situations such as its response to a physical or virtual stimulus. The future scope of this project will include imaging the brain activity on the Drosophila as it spatially navigates towards a stimulus.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER1

INTRODUCTION

Insects have known to exhibit complex behaviors [1]. Machine-insect interfaces can be used to study these behaviors in detail. It also helps in the imaging of the brain activity of these insects, which in turn will help in understanding insect cognition and neuroscience better.

Machine-animal interfaces have been studied widely in the past few decades. There have been techniques developed to both study the insect gait patterns and to develop interfaces wherein insects can control a machine in response to different stimuli. Here, we develop a machine-insect interface wherein a *Drosophila* can spatially navigate by controlling a mobile robot. This study can help in imaging the brain activity of the *Drosophila* as it navigates spatially using the mobile robot.

There have been studies on controlling the locomotion of an insect using bioelectrical interfaces. For instance, Doan and Sato [2] used electrode implants and radio waves to control the flight of a freely-flying beetle. This study created an insect-machine hybrid system. Similarly, Bozkurt *et al.* [3] created a bioelectric implant interface to trigger specific parts of the brain of a tobacco hawkmoth to control its flight. Son and Ahn [4] created a robot to study the interaction between an insect and the robot. The study used a stag beetle to get to a target and used fuzzy logic to create a learning mechanism. This learning mechanism was then used to mimic the movements of the beetle through a robot. While this is not directly focused on the machine-insect interface, it helps in learning the behavior and movement patterns of insects. Tsang *et al.* [5] experimented in making flexural neural probes to provide stimulation to the brain which can be used in locomotion controlled experiments. They created the neural probes using a carbon-nanotube enhanced material that will help in insect-machine interfaces. Ejaz *et al.* [6] used the individual cells of the visual system of a tethered *Drosophila* to create a brain-machine interface to control a mobile robot. The visual system of the fly is stimulated using a visual stimulus placed in front of the fly. Bozkurt and his group [7] presented a bioelectric interface made of micro-fabricated probes placed into insect during the growth cycle. Benvenuto and his

group [8] proposed a machine-insect interface architecture, where the insect is fixed and can be used to control drones and mobile robots to navigate through space. They hope that the proposed idea will be used for space applications to control un-manned spaceships and navigate through outer space. This can be attributed to the complex navigation skills of insects. Another method to control a mobile robot using a silkworm moth is used by Atsushi and his group [9]. The study used the signals obtained from the brain of a tethered silk moth using electrodes to control the movement of the robot. Akimoto *et al.* [10] and Tsujita *et al.* [11] have constructed robots that have been modeled based on gait patterns in insects. These studies will help in studying the characteristics of insect strides to create better and efficient robots. Asama [12] reviews the different methods available to create a cyborg insect wherein the robots and the brains of insects can be integrated together. Wessnitzer *et al.* [13] created a machine-insect interface for crickets to navigate towards a sound source. The interface uses a trackball mechanism onto which the cricket is tethered, to navigate towards the sound source. Melano [14] developed a machine-animal interface wherein the brain signals are obtained from a moth using a copper electrode and the signal obtained is used to control the rotation of a robot. Ando *et al.* [15] developed a mobile robot platform controlled by an adult silk moth reacting to odor. This odor-tracking interface used a ball tracking mechanism on which the silk moth was tethered. Kain *et al.* [16] developed a leg tracking platform for a *Drosophila* tethered to a ball. The fly's individual legs were tracked as the fly reacted spontaneously. Kanzaki's research group [17] created a model based on the odor-tracking capability of a moth and tested the algorithm on a mobile robot. Dahmen *et al.* [18] presented a design for an air-cushioned treadmill to study the walking behavior of a tethered desert ant. The design uses a hollowed-out styrofoam ball, which is air cushioned. In our study, we use a ball tracking mechanism similar to the one used by Kumagai [19] and his group in their study where they developed a robot that balances itself on a ball. The robot uses omni-directional wheels to balance on the ball. Robinson *et al.* [20] showed a mathematical representation of the velocity-level kinematics of a spherical orienting device that uses omni-directional wheels for spherical motion. The kinematics of our system will be similar to this study.

Multiple studies described above have developed passive machine-insect interfaces wherein the insect is tethered or fixed to an interface due to the lightweight of the insects compared to the equipment, which may cause biased behaviors. There is limited research that studies an active *Drosophila*-machine interface that allows the *Drosophila* to navigate. We use the Robotic Operating System (ROS) to interface between the fly tracker and the mobile robot. Our study will help in studying the behavior of a *Drosophila*. While multiple studies use virtual reality as a stimulus, we would use a live stimulus in the real world towards which the fly can navigate using the mobile robot. This study will also contribute, in the future, to studying the brain activity of the fly as it spatially navigates using the mobile robot. This research study is aimed at developing a machine-insect interface to achieve spatial navigation by a freely walking *Drosophila* without tethering. This will prove to be useful for further studies on brain activity imaging of the *Drosophila*'s brain.

CHAPTER 2

SYSTEM DESIGN

1 Overview

In this study, we developed a fruit fly (*Drosophila melanogaster*)-operated mobile robot interface, which enables us to interpret the animal's behavior. The schematic drawing of the system is shown in fig.1. The system consists of two major sub-systems: the Active Omni-directional Treadmill (AOT) and the mobile robot.

The AOT comprises of the three omni-directional wheels which are oriented at an angle and at a distance from the center such that the transparent sphere rests on all three wheels. The fly that is housed in the acrylic chamber is allowed to walk on top of the ball and the position of the fly is tracked using the near-infrared imaging system and the image processing system. The Near Infrared (NIR) camera is placed directly below the fly and images the fly. The omni-directional wheels help in producing a counter-motion which resets the fly back to its original position. The counter-motion produced is based on the velocity commands computed using the algorithm described in section 3.1 . The interface is controlled by the image-processing system which is programmed in the host system. The mobile robot moves separate from the interface reproducing the motion produced by the AOT motors.

The mobile robot has a microprocessor which computes the control input velocity for the motors in the mobile robot. The servo motors of the mobile robot receive commands from the Robotic Operating System (ROS). The microprocessor on the mobile robot also enables us to run ROS, a program which allows interaction with multiple components within the system. In this system, it has three different nodes which communicate with each other via a central module called roscore. The image processing system is one of the nodes that communicate with the mobile robot. It also processes the images obtained from the NIR camera and computes the position of the fruit fly. This system can be integrated with other techniques (e.g. brain imaging) easily as it can hold the animal position in one place while it is moving.

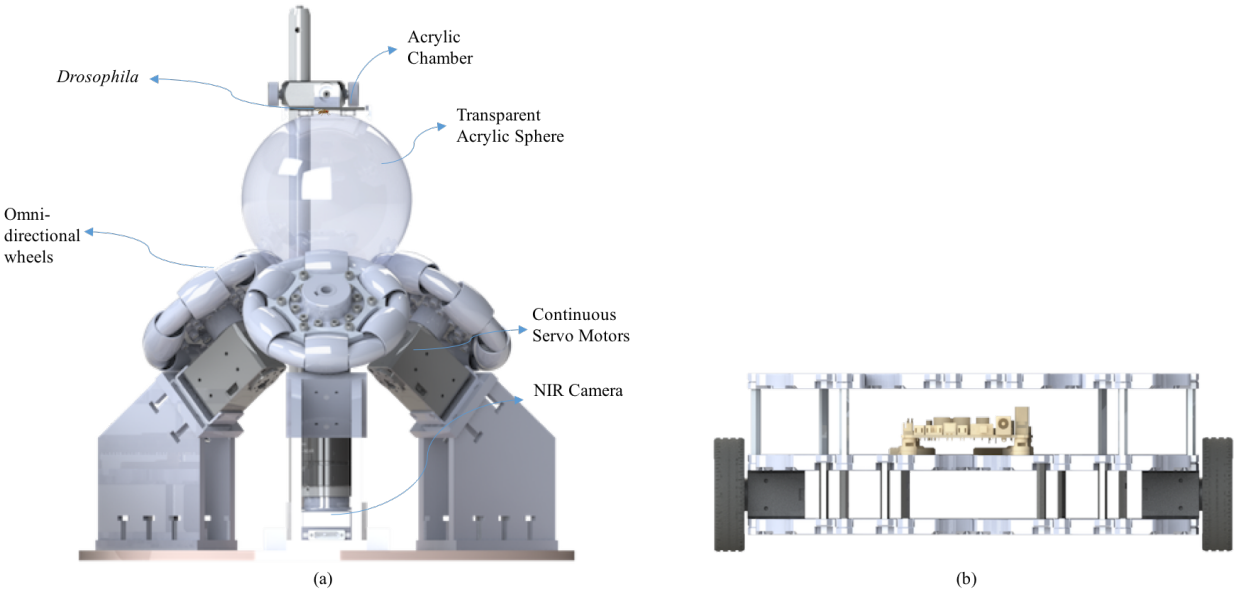


Figure 1: **Front view of (a) Active Omni-directional Treadmill (b) Mobile Robot**

The ROS architecture for our system is shown in fig.2. There are three main nodes of the system, which interact with each other. ROS works on a 'publisher-subscriber' architecture which contains nodes acting in sync with one another via a central core unit called the Roscore. All publishers publish messages (commands) to the Roscore and all subscribers that are subscribing to a topic published by a node receive the messages from the Roscore. In fig.2 the mobile robot tracker node is a subscriber only and it receives just the positions from the robot as it moves. The image processing node is both a subscriber and a publisher. It receives the images from the camera node, which publishes the images as raw data. The image processing node computes the velocity data and publishes to the roscore, which is obtained by the mobile robot.

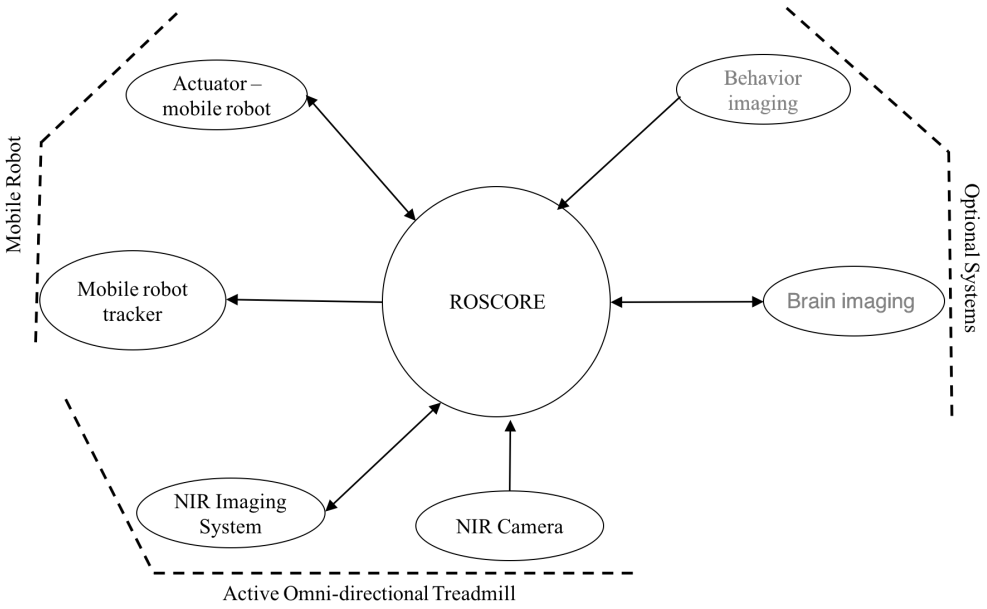


Figure 2: **ROS architecture for our system**

2 Control Design

This section describes the control design of the system. Fig.4 depicts the control block diagram for the setup described in section 1. As the fly walks on top of the acrylic sphere, the images captured by the NIR camera are processed using the program described in listing 1 of the appendix to obtain the centroid position of the fly. The controller 1 computes the control input based on the position error between the fly position and the origin. Fig.3 shows the fly being imaged by the NIR camera. The red box shown in the image is the tolerance of 50 pixels, which corresponds to about 1.3 mm in distance from the origin. The red circle in the image is the detected centroid of the fly.

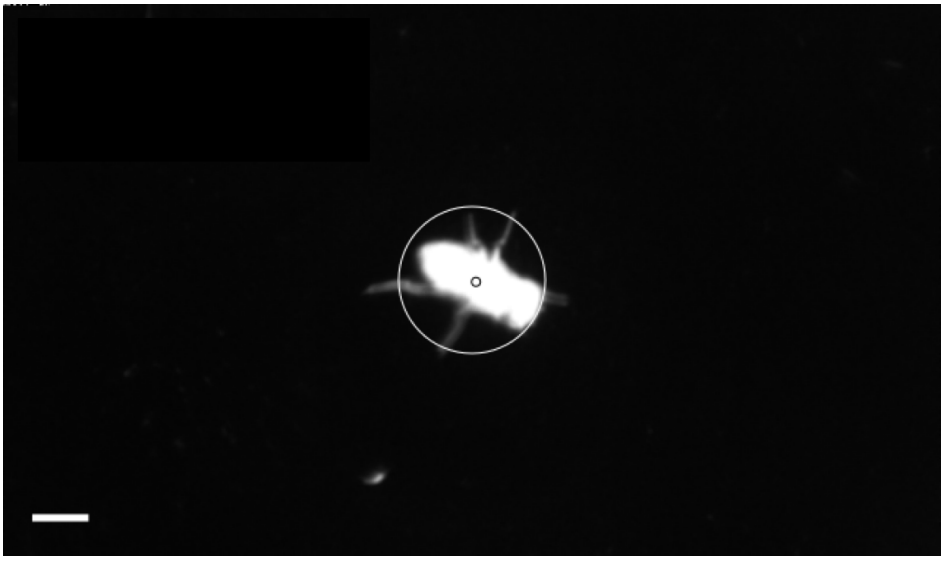


Figure 3: **A single frame of the system compensating the fly motion. The yellow box represents the tolerance region within which the system resets the fly. The scale bar indicates 1mm length**

The AOT resets the fly in this region with every movement of the fly. This means that whenever the fly walks on the ball, the ball rotates to reset the fly to the origin. This is achieved using the omni-directional wheels to create counter-motion. The position of the fly is used to determine the rotation needed for the ball. As the fly walks on the ball, the error, Δx, Δy, and $\Delta\theta$ is computed to obtain the compensation error needed. Based on these values, the angular velocities of the AOT motors, ω_1, ω_2, and ω_3 are computed.

$$X_{fly} = \int \Omega_{sphere} dt + \epsilon \tag{1}$$

Eq. (1) can be used to determine the position of the fly, X_{fly}. Ω_{sphere} is the angular velocity of the sphere and ϵ is the error computed based on the fly's movements.

The position of the fly is also used to move the mobile robot remotely. The actuators in the mobile robot are servo motors, which will receive velocity commands to move the mobile robot based on the fly movements. The mobile robot reproduces the motion of the sphere.

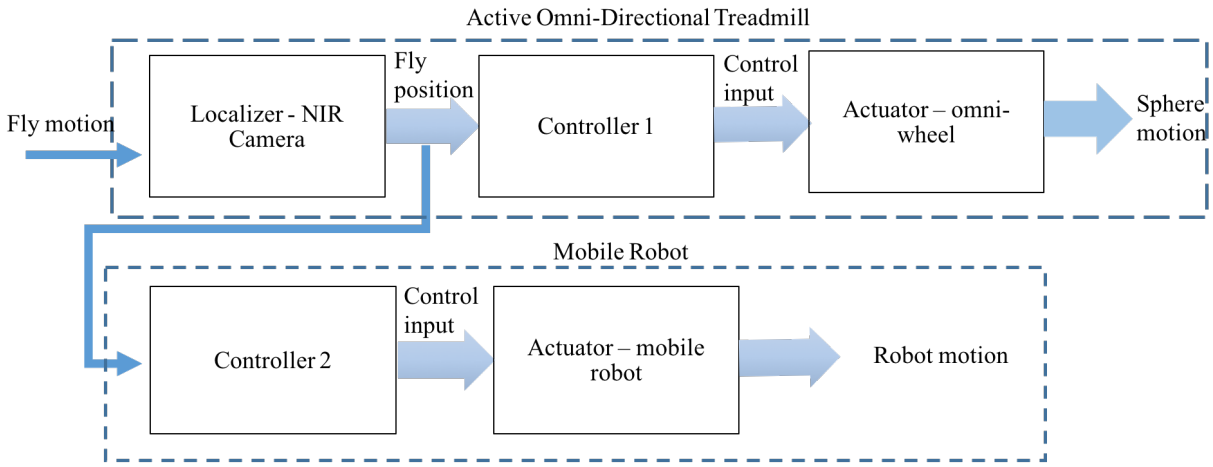


Figure 4: **Control block diagram of the system**

3 Active Omni-directional Treadmill (AOT) Subsystem Design

The AOT mainly consists of the transparent sphere on which the fruit fly walks and three omni-directional wheels operated by servo motors. The continuous servos contain contactless absolute encoders which enable us to receive positional feedback from the servo. The AOT acts as an interface between the fruit fly and the mobile robot and is connected to the host system and communicate via serial communication. The AOT consists of an acrylic plastic ball of 4-inch diameter, which rests on three omni-directional wheels oriented at a 40° angle to the axis of the ball. To prevent the fly from flying off the ball, we cover it with a clear acrylic chamber, which is shown in fig.5. The enclosure shown in the image is an acrylic chamber bounded by a window in the front and covered on top with a glass slab. The glass slab is bonded to the acrylic using using layered Polydimethylsiloxane (PDMS) sheets. The window is made of acrylic as well and is fixed using acrylic adhesive.

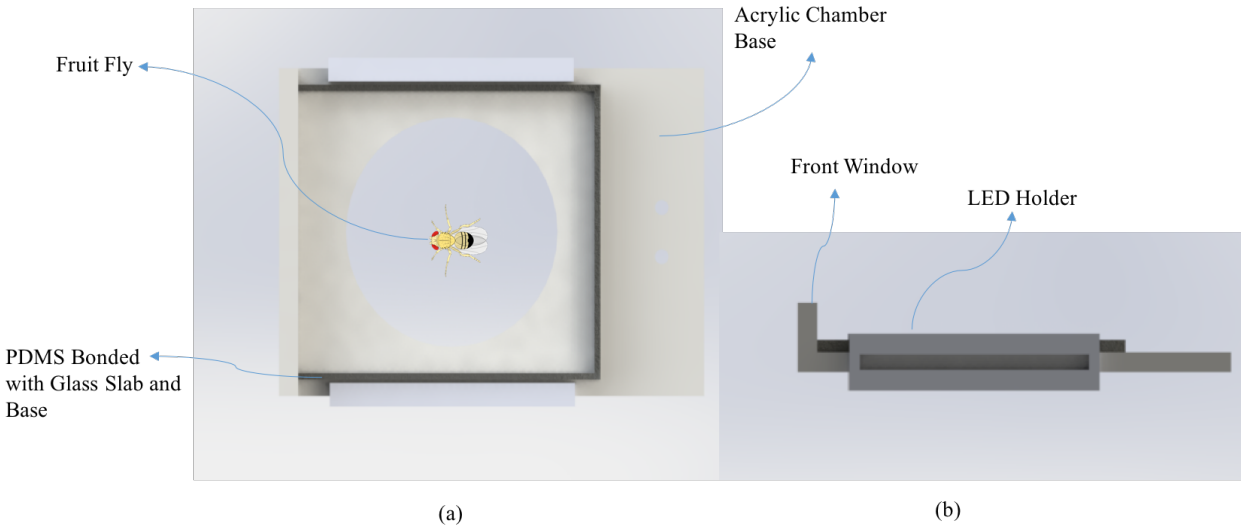


Figure 5: **(a) Top view of the acrylic chamber (b) Side view of the acrylic chamber**

The acrylic chamber also has LEDs on either sides that internally reflect through the base plate of the housing. The LEDs are positioned such that the light scattered from the fly will fall on the camera's sensor. The position of the LEDs are shown in fig.5. The LED holders are designed to hold 12 LEDs each. Twenty four LEDs (twelve on each side of the acrylic housing) have been used to provide sufficient illumination for the imaging of the fly. The circuit diagram for the LED circuit is shown in fig.6. Each of the array of 12 LEDs consist of two sub-arrays of six LEDs connected in parallel. Each LED is rated a forward voltage of 1.5V and a forward current of 100mA. We used a 33Ω resistor for the 24 LEDs based on the voltage and current these draw.

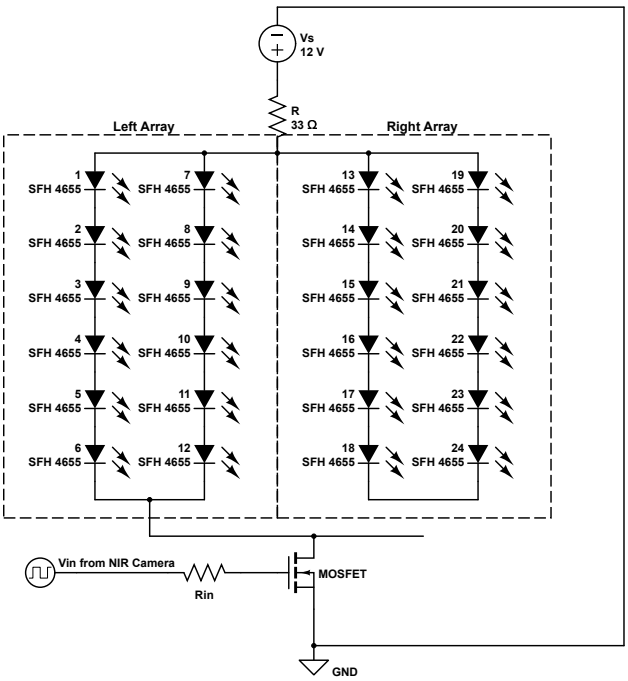


Figure 6: **Circuit diagram for the LED setup in the system**

Fig.7 shows the omni-directional wheels used in this system. These wheels are 100mm in diameter and are made of aluminum [21]. The omni-directional wheels had to be carefully selected so that all the rollers are located on the same radius. Any difference in height between the rollers meant that the acrylic ball would not roll on the wheels smoothly. As is can be seen in fig.7, the distance between the rollers is almost none. To ensure a smooth rotation of the ball, the omni-directional wheels shown in fig.7 were chosen.

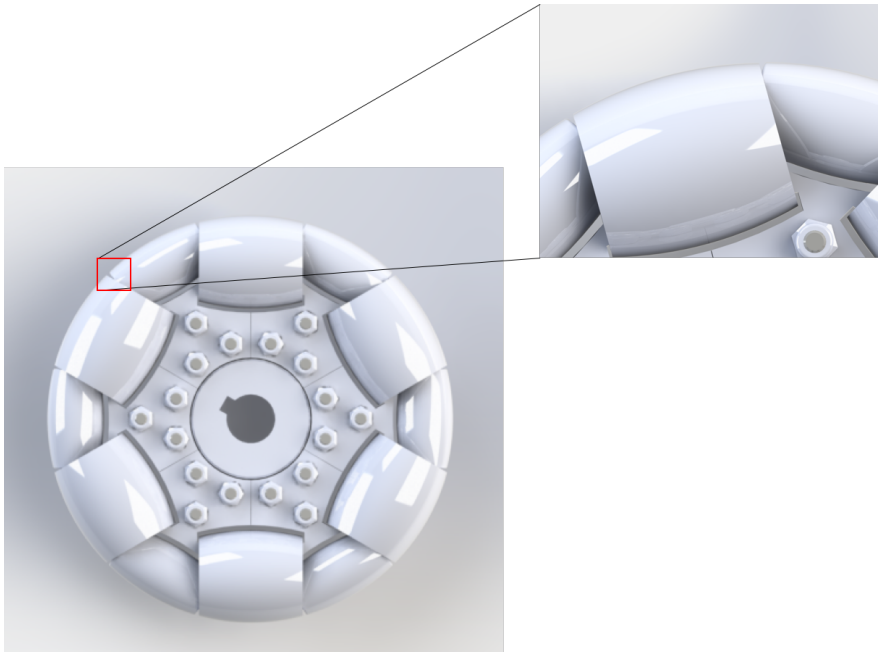


Figure 7: **Omni-directional wheels used in the AOT**

These are similar to the ones used by and his group to create the ball-balancing robot [19] . The omni-directional wheels had to be strategically placed from each other so that the ball is centered with the camera. A custom servo horn shown in fig.8 was designed and 3D printed to attach the omni-directional wheels to the selected servo motors.

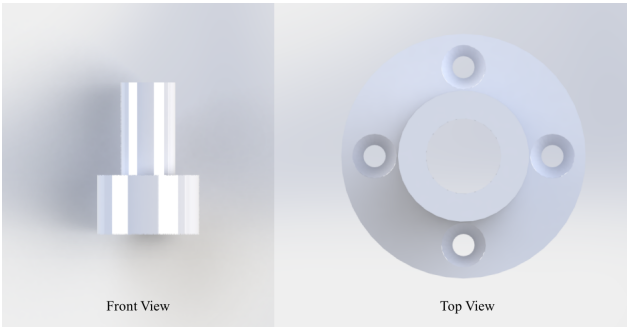


Figure 8: **3D printed custom servo horn for the omni-directional wheels**

The motors used in this system are continuous servo motors, which enable us to control the position of the fly via the rotation of the acrylic ball. Three servo motors placed at 0°, 120°, and 240° respectively. The three servo motors are connected to each other via Daisy Chain connections using TTL cables. One of the motors is connected to the micro-controller (OpenCR)

and pass on the power and signal to the other two motors. Fig.9 shows the overall system connection with the host system.

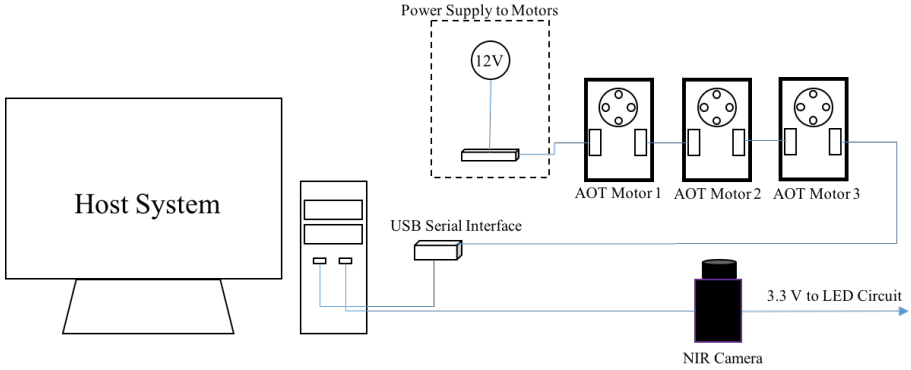


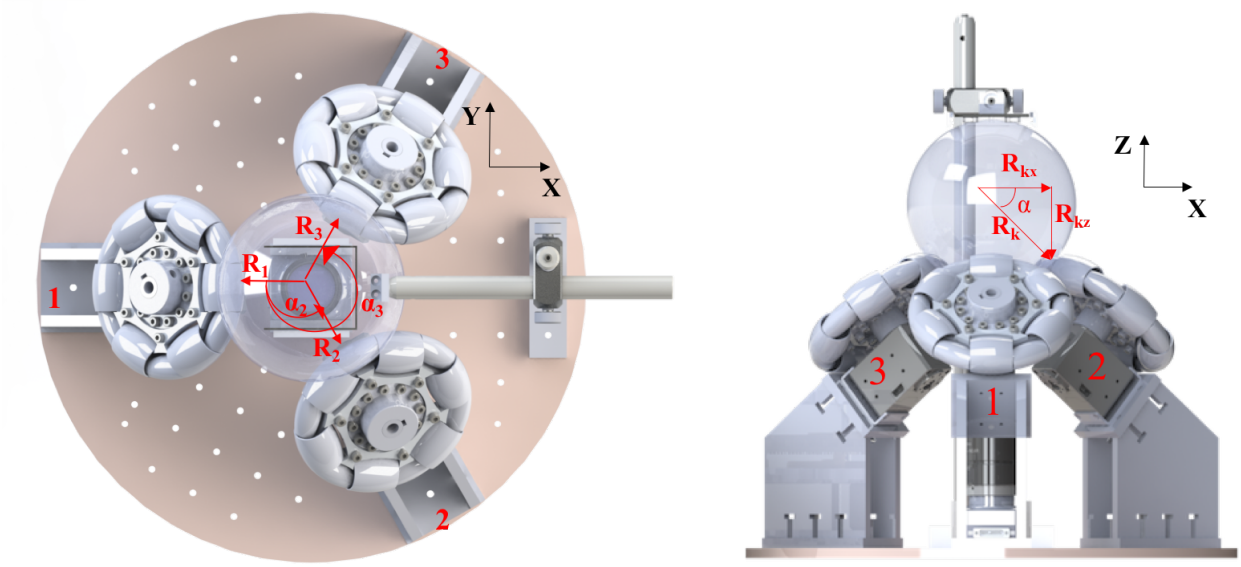
Figure 9: **Host System-AOT Schematic**

This system uses a NIR camera to capture the images of the fly. The camera utilizes a infrared bandpass filter to selectively pass light of wavelength 850nm. This helps by preventing other objects in the field of view of the camera in the visible light region from being captured in the image. Since, infrared LEDs are used in the system, the light scattered by the fly will be captured by the camera.

3.1 Kinematics of Active Omni-directional Treadmill

In fig.10, the ball resting on the omni-directional wheels are resting on a roller on each wheel. The wheels are arranged in a triangular setup to provide stability. The geometric transformation from the omni-directional wheel coordinate system to the global coordinates is given by [20]:

$$T_i = \begin{bmatrix} \cos\alpha_i & \sin\alpha_i & 0 \\ -\sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2}$$



Top View Side View

Figure 10: **Top view and Side View of the AOT setup**

The angle α_i is the angle between the axis of the wheels (located at 0°, 120°, and 240°) and i corresponds to the i^{th} wheel. Eq.(2) can be used to convert the angular velocities of the sphere to the global coordinates. Our objective is to obtain the velocity to reset the fly back to the origin.

$$\begin{bmatrix} r\omega_y \\ r\omega_x \\ \omega_z \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{3a}$$

Where ω_x, ω_y, and ω_z are the three vector components of the angular velocity of the sphere, r is the radius vector of each omni-directional wheel, and $\dot{x}$, $\dot{y}$, and $\dot{\theta}$ are the velocities of the fly in the Cartesian coordinate system, assuming that in the area where the fly is walking on the sphere, the curvature of the sphere is extremely small. Ideally, the omni-directional wheels are identical and no slip exists between the omni-directional wheels and the sphere. Using eq.(3a), we can obtain the angular velocity of the omni-directional wheels as well.

$$\begin{bmatrix} \omega_y \\ \omega_x \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{r} & 0 \\ \frac{1}{r} & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = J \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3b}$$

Where J is the Jacobian,

$$J = \frac{R_{1_z}}{|R|^2} \begin{bmatrix} R_{1_z} & R_{1_z}\cos\alpha_2 & R_{1_z}\cos\alpha_3 \\ 0 & -R_{1_z}\sin\alpha_2 & -R_{1_z}\sin\alpha_3 \\ -R_{1_x} & -R_{1_x} & -R_{1_x} \end{bmatrix} \tag{3c}$$

and R_k ($k \in 1, 2, 3$) is the vector from the center of the sphere to the point of contact on the omni-directional wheel as shown in fig.10. Therefore, eq.(3b) can be re-written as:

$$J^{-1} M \dot{X} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3d}$$

where, $M = \begin{bmatrix} 0 & \frac{1}{r} & 0 \\ \frac{1}{r} & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$.

Eq.(3d) can be used to obtain the velocity input required for the servos to rotate the omni-directional wheels.

3.2 Localization of Fruit Fly

When the light interacts with an object, it reflects off the object and produces an image when it hits a detector such as a camera. This is known as bright-field imaging. At a certain angle of incidence, the light does not impinge on the object and creates a dark background. The light collected by the sensor is no longer reflected off the object but rather scattered or refracted off the object. This helps us capture images of particular interest. This is called the dark-field imaging technique. We use this technique to image the fruit fly and locate its position. The LEDs placed

on the side of the the acrylic chamber provides the illumination that is scattered by the fruit fly in the dark-field. This scattered light is captured by the NIR camera and is processed by the imaging system, which works based on blob image processing, to obtain the centroid of the fly. To create the pulsing mode of the LEDs, we use a MOSFET transistor as a switch and the pulse signal output from the camera as the input signal for the LEDs. The NIR camera gives a output pulse signal with capture of every image. This enables the pulsing mode for the LED. Fig.6 shows the circuit used to create the pulsing mode. To narrow the image to the range of the fly's motion, we chose a frame height and width of 376 pixels and 640 pixels respectively. Fig.11 shows the image size and the origin of the image.

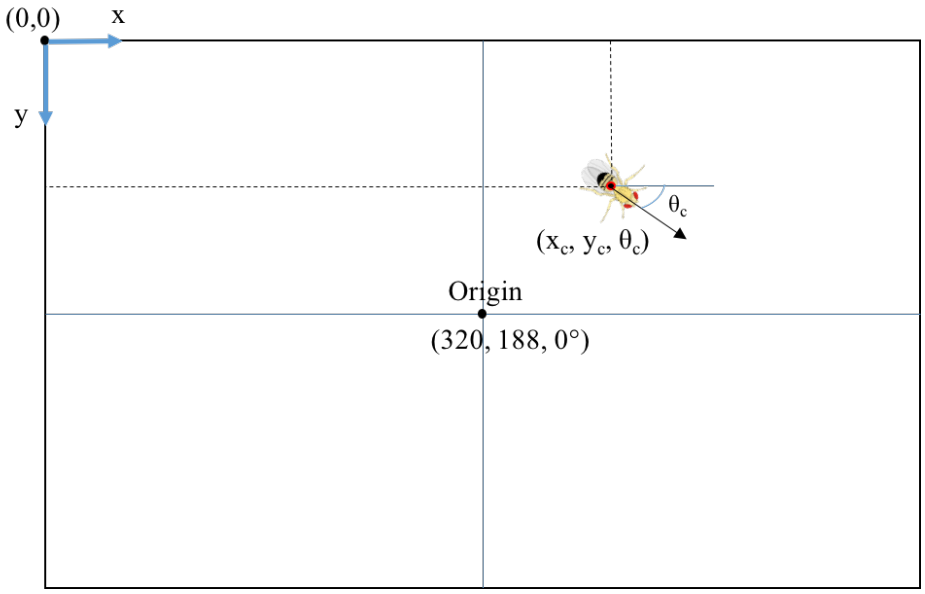


Figure 11: **Image region of interest**

This converts to a width of 16.67mm and a height of 9.86mm in the xy plane on which the fly walks. The size of the image in standard units of measurement was found by placing an object of known height and width in the image frame and the conversion was found to be 1mm per 38.4 pixels. The origin where the fly is to be reset every time is the center of the image which is $O = (w_{im}/2, h_{im}/2)$ where w_{im} and h_{im} are the width and the height of the image respectively.

This was used to calculate the error between the fly's position and the origin.

$$\Delta x = x_c - p_{target,x} \tag{4a}$$

$$\Delta y = y_c - p_{target,y} \tag{4b}$$

$$\Delta \theta = \theta_c - p_{target,\theta} \tag{4c}$$

Where Δx, Δy, and $\Delta \theta$ are the inputs to compute the velocities, x_c, y_c, and θ_c are the x-coordinate, y-coordinate, and the orientation of the centroid of the fly, and $(p_{target,x}, p_{target,y}, p_{target,\theta}) = (w_{im}/2, h_{im}/2, 0)$.

The program described in listing 1 of the appendix is used to obtain the position of the centroid of the fly. Based on this, the program described in listing 1 in the appendix calculates the angular velocity of each motor to reset the fly back to the original position. This controller program uses the algorithm described in section 3.1.

4 Mobile Robot Kinematics

The Waffle is a mobile robot, which supports the Robotic Operating System (ROS), which allows easy integration of other systems. This is one of the main motivation for us to use the Waffle for this study. The servos of the robot receive the velocity commands based on the position of the fly. The mobile robot moves based on the movement of the fly. Fig.12 shows the kinematic model of the mobile robot. The general coordinate of the mobile robot is described as:

$$\mathcal{X} = \begin{bmatrix} x_m \\ y_m \\ \theta_m \end{bmatrix} \tag{5a}$$

Where, x_m and y_m are the positions in the Cartesian coordinate, and θ_m is the orientation. Each individual wheel of the robot contributes to its motion. The velocity of the robot is

defined as the average velocity of the right, V_r and the left wheel, V_l of the robot.

$$V_m = \frac{(V_l + V_r)}{2} = \frac{(\omega_l + \omega_r)r}{2} \tag{5b}$$

Similarly, the angular velocity of the robot, is determined by the following equation,

$$\dot{\theta_m} = \frac{(V_r - V_l)}{L} = \frac{(\omega_r - \omega_l)r}{L} \tag{5c}$$

Where, r is the radius of the wheels.

Resolving the components of the velocity, we can obtain the kinematic model of the mobile robot as:

$$\dot{\mathcal{X}} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{5d}$$

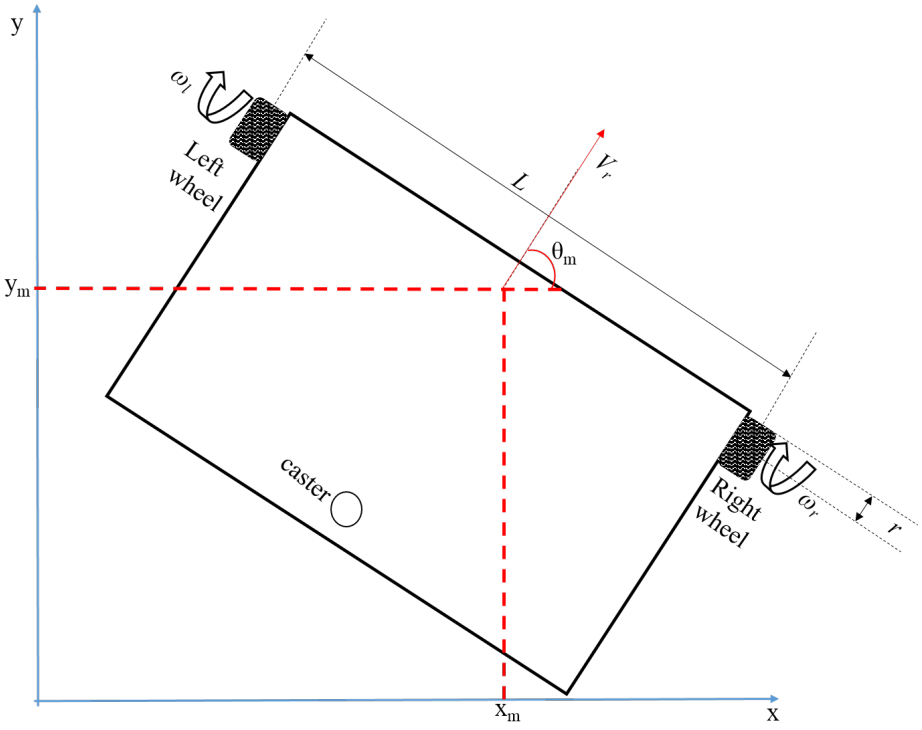


Figure 12: **Kinematic model of the mobile robot**

5 Software Design

5.1 Low-level Software Development

The low-level software programs were implemented on the micro-processor on the robot. The programs were written for the micro-processor were written based on the Arduino Integrated Development Environment (IDE) language. The algorithm described in section 3.1 was used to calculate the control input velocities, ω_1, ω_2, and ω_3 for the three motors in the AOT. This is one of the nodes that is a part of the system's ROS architecture. This node receives commands from the image processing node, which published position data of the fly. This node converts these position messages into velocity commands for the motors of the AOT and the robot. The program, which describes this node can be found in listing 1 of the appendix.

5.2 High-level Software Development

The high-level software development was programmed on the host system. This system is where the NIR camera is connected to as well. The host system is responsible for running the roscore for the mobile robot and the camera node. The image processing node subscribes to the camera node and processes the image obtained to extract the position of the fly. In this system, the image processing is done using OpenCV library. The image processing node is launched along with the camera node. The launch file described in listing 3 in the appendix shows the different settings such as the frame rate, shutter speed, gain, strobe duration etc that is being configured when the node is being launched. This allows us to selectively configure the image to our region of interest (roi). Along with the camera node, the image processing and the mobile robot tracking node are launched so that as the images are captured, the images are subscribed by the image processing node and the image is processed to obtain the centroid. The image moment of the image is used to compute the centroid of the contour detected. In this case, the detected contour is the fly. The program is written so that only the largest detected contour in our roi is considered for processing. The image moment is a weighted average of the pixel intensities in an image. Using

the ”Hu Moment” function of OpenCV library, the image is processed to obtain the centroid coordinates x_c, y_c, and the orientation, θ_c of the centroid fly.

CHAPTER 3

RESULTS AND DISCUSSION

The program to reset the position of the fly back to the origin was written on the host system. Tests run on the system showed that there was a tolerance needed for the compensation. The reason for this was that the servo motors used in the AOT did not have smooth torque control at low velocities. At low velocities, there was jerk associated with the movement of the wheel and this caused delay in producing the necessary counter-motion to reset the fly. Hence, a tolerance of 50 pixels or 1.3 mm in both the x and y directions from the center of the image and a tolerance of 3 degrees was set for the theta compensation. Fig. 13 shows six frames with fly movement and compensation. As it can be observed, the red box at the center of the image is the tolerance of 50 pixels around the center of the image. The red circle on the fruit fly is the centroid of fly being reset back within the tolerance limit. The frame number and the x, y, and θ (measured in radians) of the centroid is also displayed in each frame. It can be observed that as the system tries to reset the fly, the centroid coordinates gets closer to the origin of (320,188,0).

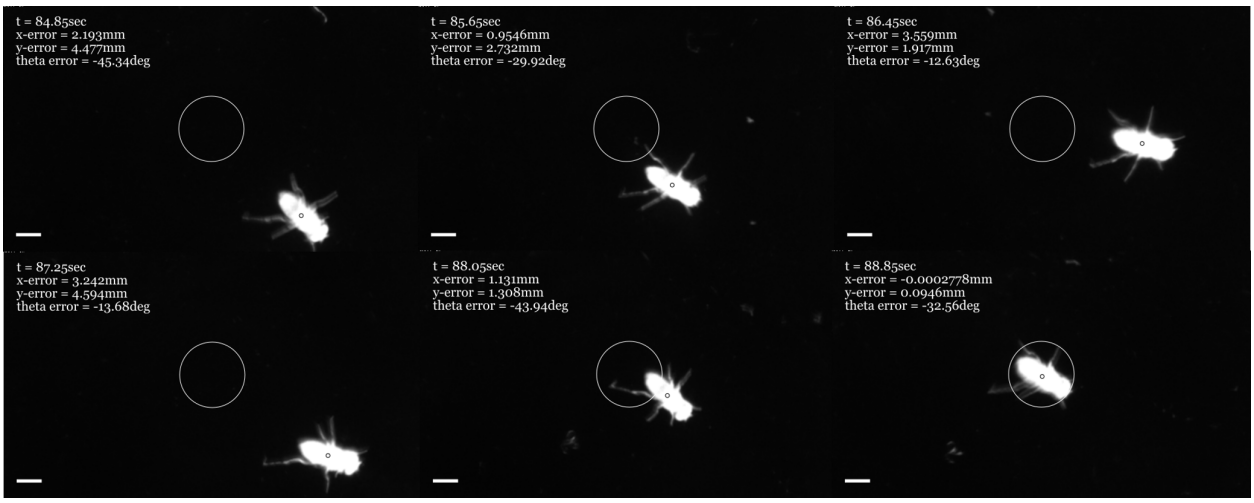


Figure 13: **System resets the fly back to the center of the image. The scale bar indicates 1mm length**

As the fruit fly moves on the sphere, an error between the origin and the centroid of the fly is obtained. Using this, the error can be computed. The error distance is computed using eq.(6)

$$d_{error} = \sqrt{x_{err}^2 + y_{err}^2} \tag{6}$$

Where, the d_{error} is the distance the fly moves before compensation, x_{err} is the difference between the x-coordinate (320 pixel) of the center of the image where the fly will be reset to and y_{err} is the difference between the y-coordinate (188 pixel) of the center of the image where the fly will be reset to. The average error of the distance was found to be 1.821mm. This means that the fly moved a distance of about 1.821mm before being reset to the origin within the tolerance. Fig.14 shows how the distance error changes during experimental time and the corresponding linear velocity of the sphere. It can be observed that when there is a large distance error, the velocity of the sphere rapidly increases to compensate the motion of the fly. This can be observed at around 70 seconds for instance.

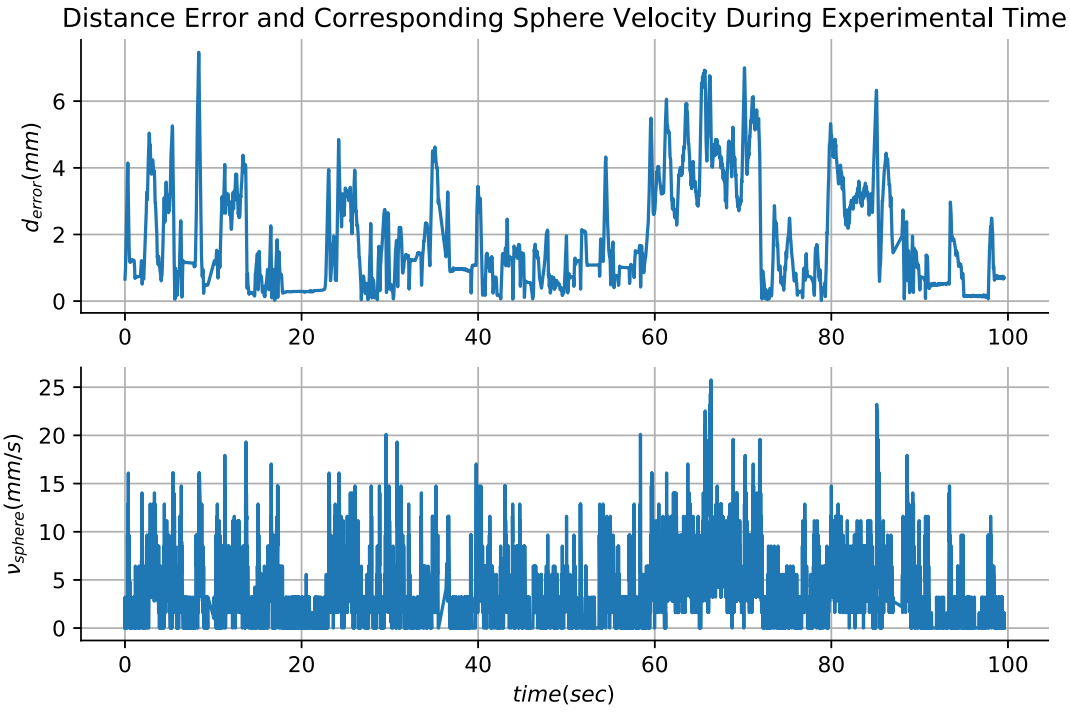


Figure 14: **Distance error during experimental time and the corresponding linear velocity of the sphere**

After the motor compensates the motion of the fly, it comes to rest before moving again

for the next compensation. This can also be observed in fig.14. Therefore, the motor has to overcome the stiction before reaching the desired velocity. The time the motors take to over come the stiction and start compensating for the error after image processing is about 110 milliseconds. This can be better observed in fig.15.

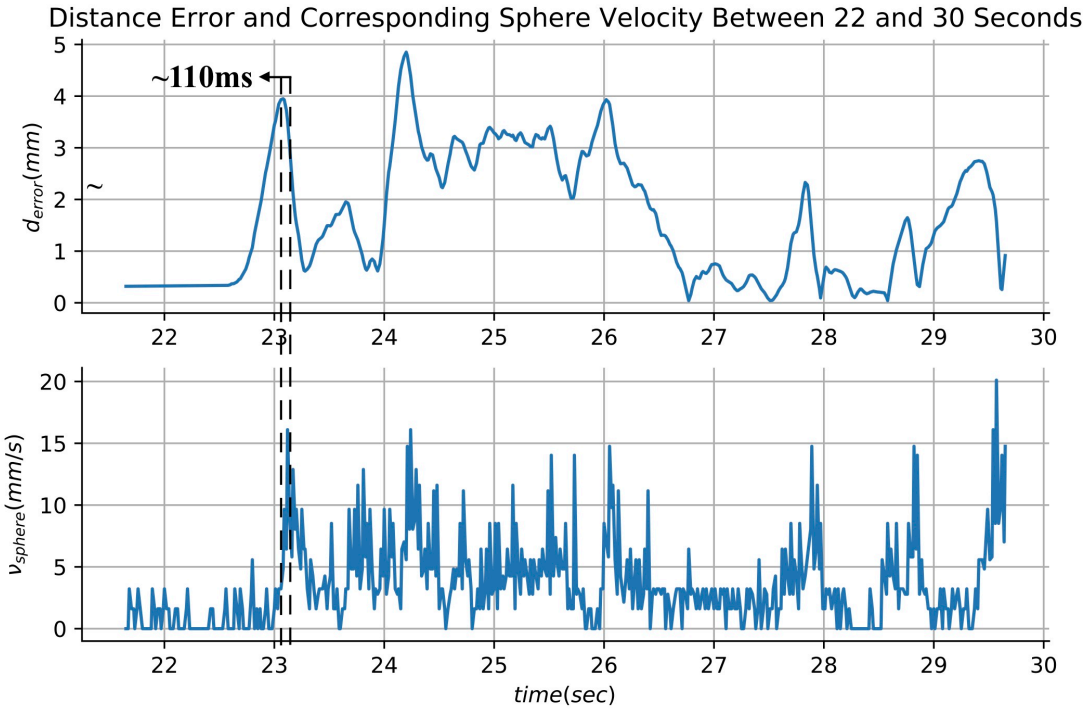


Figure 15: **Distance error and the corresponding linear velocity of the sphere between 22 and 30 seconds. The motors move approximately 110 milliseconds after the image processing is done.**

Similarly, the θ_{error} of the system is computed as the amount of rotation needed for the sphere to align the fly with the x-axis or zero radians. The average θ_{error} of the system was found to be 7.039° . This means that, the fly rotates about 7.039° before the motor starts to compensate for its rotation and resetting it back to the center of the image. Fig.16 shows the θ error during experimental time and the corresponding angular velocity of the sphere about the z-axis. The sphere's angular velocity changes as the fly's rotation increases. This indicates that the system is compensating for the angle error well within the tolerance.

Figure 16: θ **error during experimental time and the corresponding angular velocity of the sphere**

Fig.17 shows a closer look at the angular velocity of the sphere with respect to the fly. As it can be seen, the sphere's angular velocity changes, although with a delay, to compensate the motion of the fly.

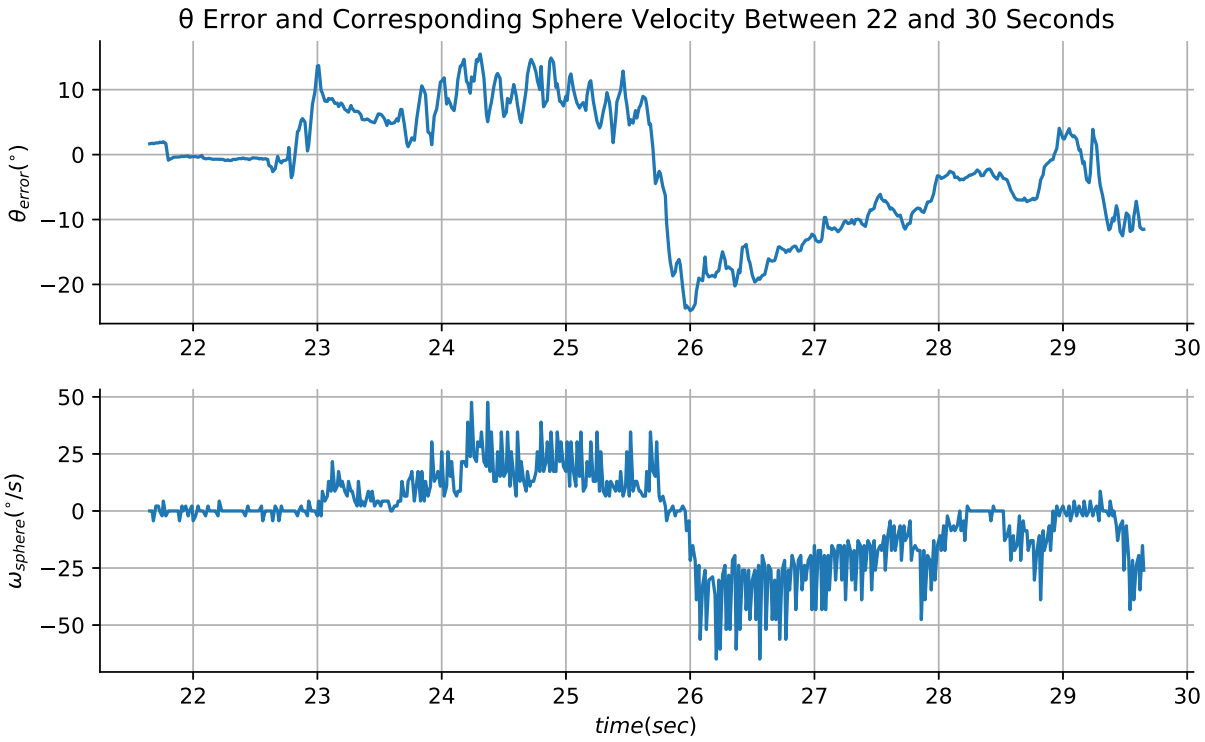


Figure 17: θ **error during experimental time and the corresponding angular velocity of the sphere between 22 and 30 seconds**

The input and the output angular velocities of the AOT motors were computed as well. Fig.18 shows the how the input and output velocities of the motors change to compensate the fly's movements. Fig.19 shows the input and output velocities between 22 and 30 seconds of the experiment. The total experimental time is 100 seconds. It can be observed that the output velocity is noisy. This can be attributed to the low resolution of the encoder in the motors providing feedback.

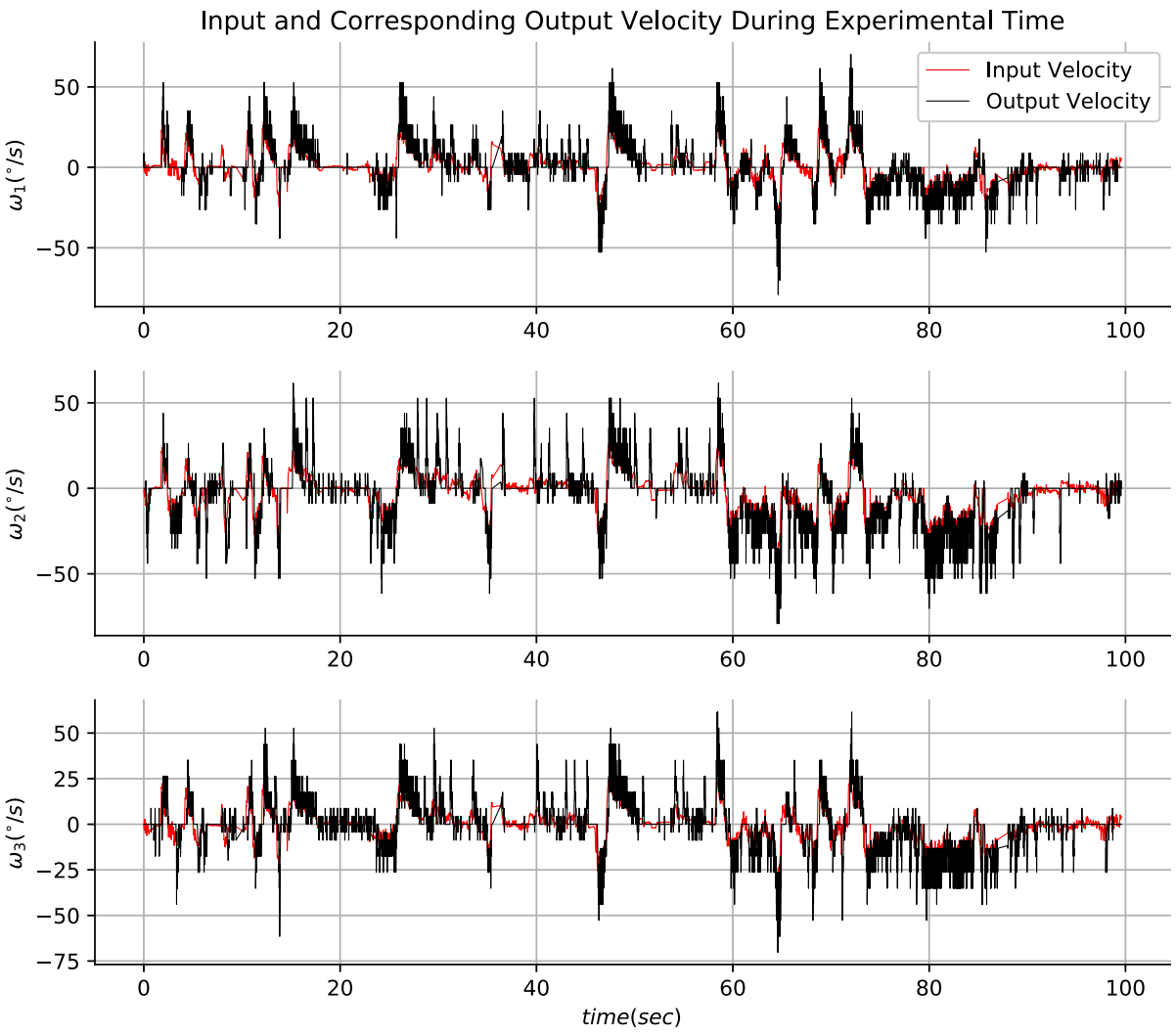


Figure 18: **Input and output velocity of the three AOT motors during the experimental time**

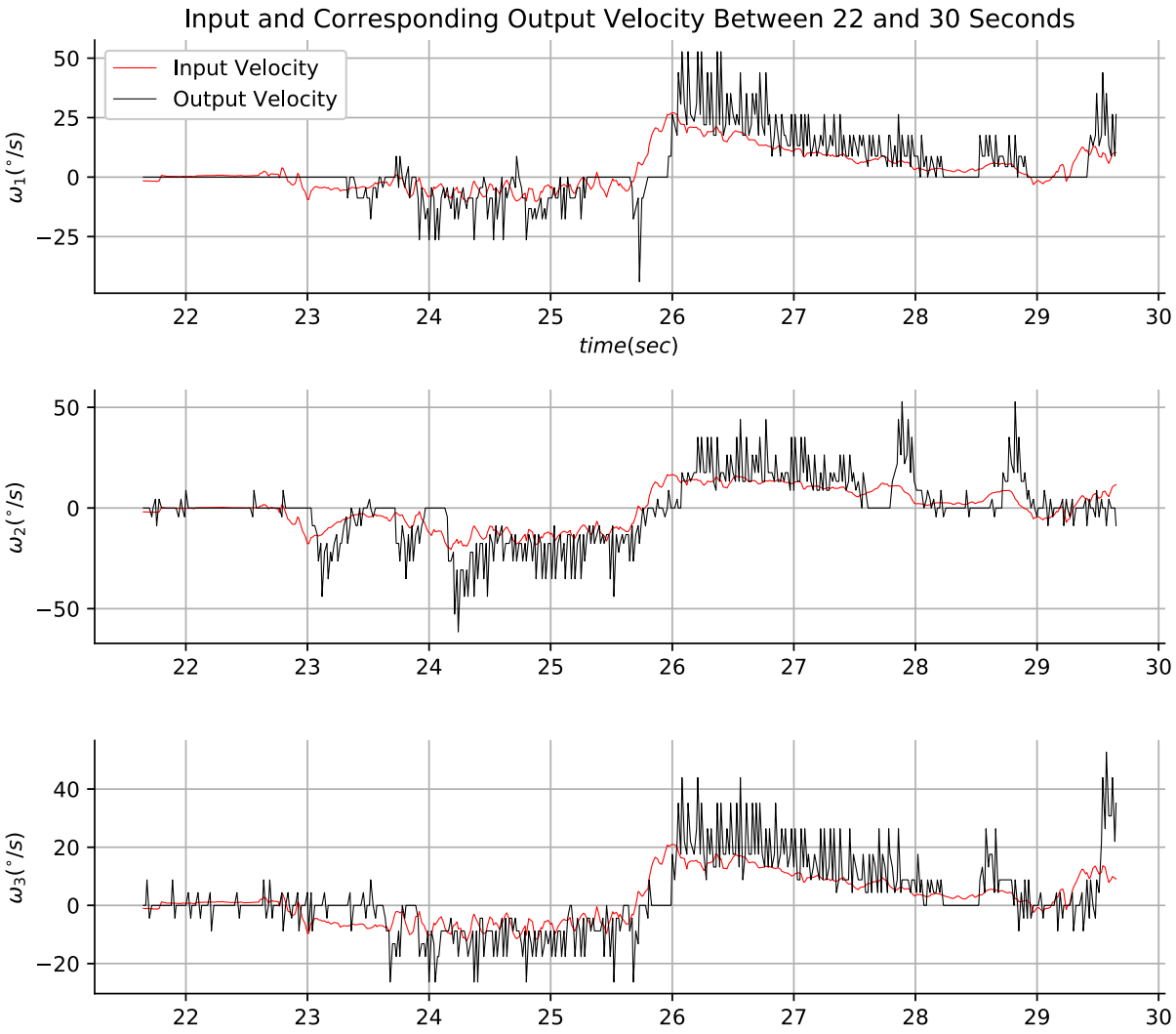


Figure 19: **Input and output velocity of the three AOT motors between 22 and 30 seconds. The noise in the output velocity data can be attributed to the low resolution of the encoders.**

In this system, the mobile robot was programmed to move like the sphere. The velocities of the mobile robot motors were sent after image-processing and the position of the robot was obtained by reading the encoder values of the right and the left wheel of the robot. Fig.20 shows the path taken by the sphere in this system. This was computed by accumulating the position of the sphere based on the position data from the AOT motors. The encoders in the motors enabled us to obtain the position feedback.

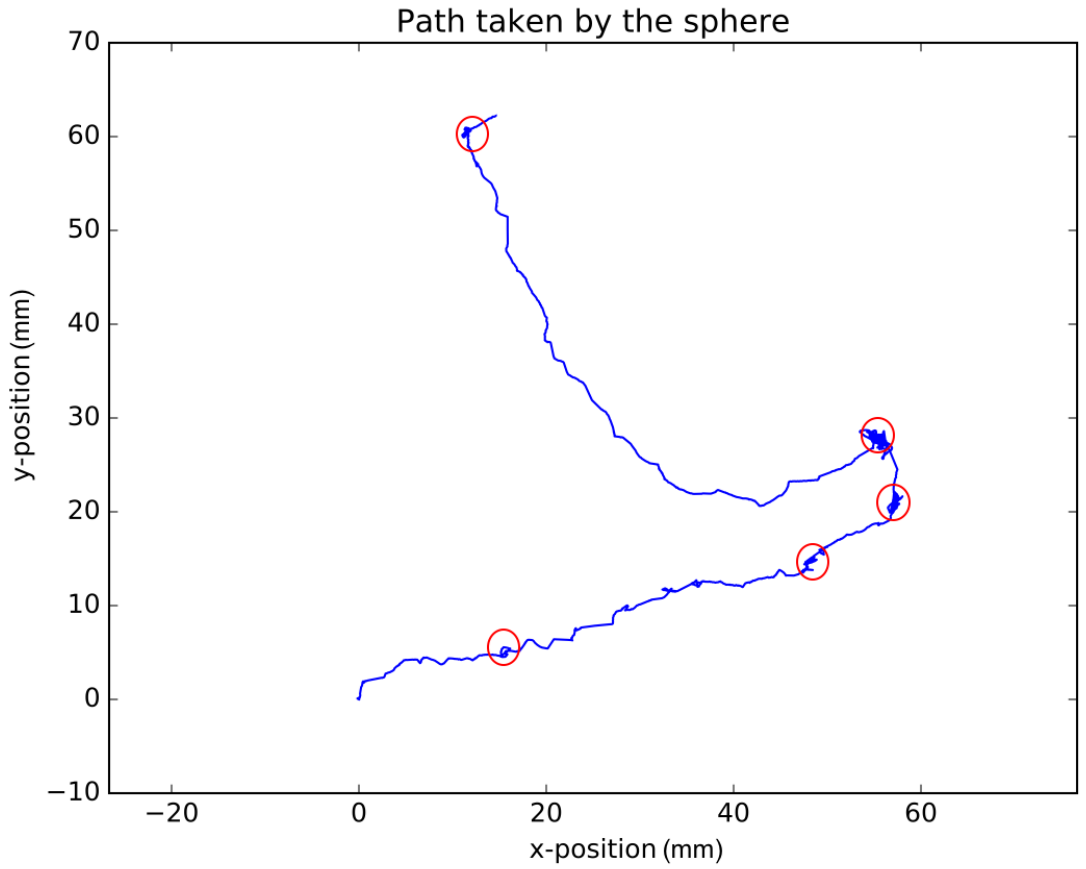


Figure 20: **Path taken by the sphere in compensating the motion of the fly**

Since, the mobile robot was given the same commands as that of the sphere, the path of the mobile robot was plotted as well and is shown in fig.21. The similarities in the path taken by the sphere and the mobile robot can be observed from the two plots. The red circles depicts the changes in direction in both the sphere and the mobile robot. It can be seen that the mobile robot does not switch directions as much as the sphere. This can be attributed to the low gain set for the motors of the mobile robot. To obtain better control, the motor's gain can be set higher to match the sphere's motion better. This will be a part of the continued work that will be done in the future as the scope of the system increases as well.

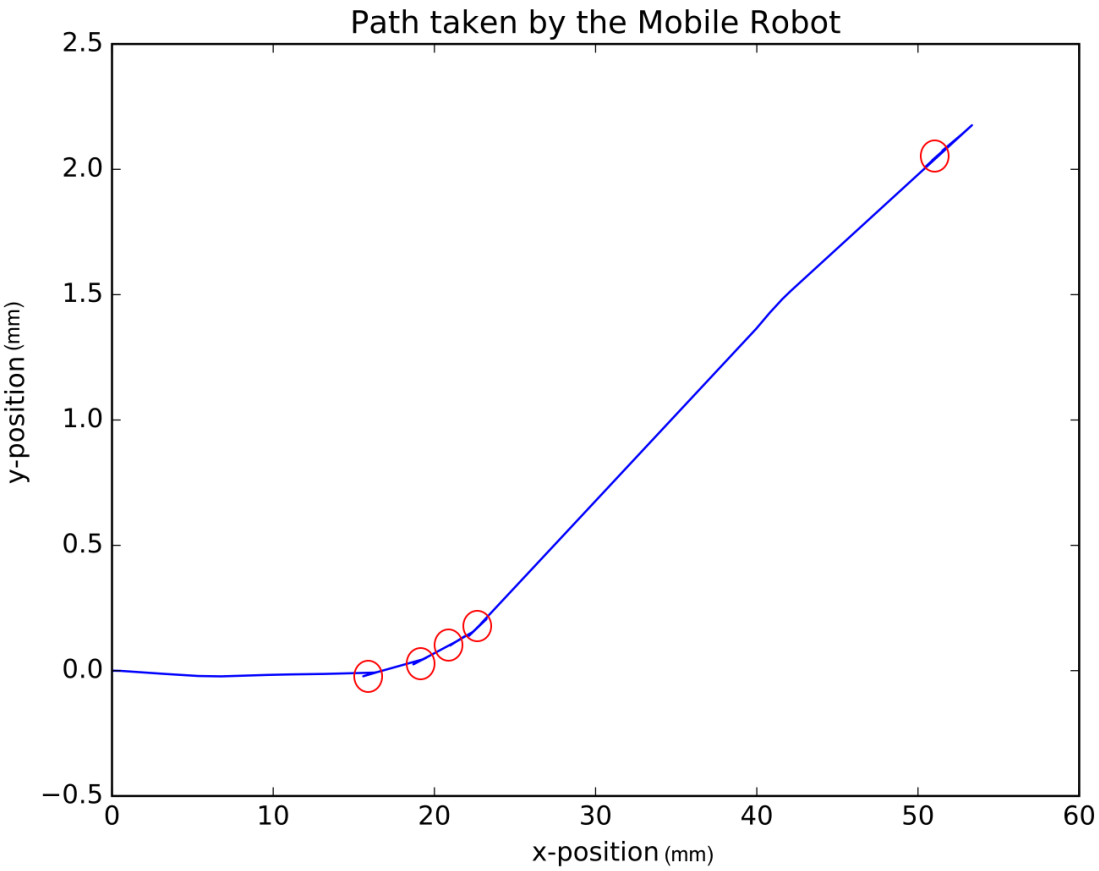


Figure 21: **Path taken by the mobile robot as it navigates spatially**

CHAPTER 4

CONCLUSION

The development of a fruit fly-mobile robot interface was described. The system consists of the Active Omni-directional Treadmill (AOT) which consists of an acrylic sphere that acts as the treadmill for the fruit fly on top of the sphere. As the fly walks on the ball, the motion of the fly is compensated by creating counter-motion. The kinematics of this AOT was described in section 3.1 in chapter 2. The motion of the sphere is also reproduced in the mobile robot, which navigates spatially. The mobile robot kinematics used is described in section 4 in chapter 2. The image processing node processes the images captured by the NIR camera to compute the velocity needed for the motors to compensate the fly's movements.

Chapter 3 described the experimental results obtained. The average distance (d_{error}) the fly moved before being compensated was found to be 1.821mm and the average orientation change (θ_{error}) the fly made before compensating was found to be 7.039° . The major limitation of the system is the delay in compensation of the fly's movements. The delay was found to be approximately 110 milliseconds between the end of image processing and motor's movements. The future scope of this project will cover using motors in the system with better torque control. This will reduce the delay significantly as it can overcome the stiction faster. The mobile robot's movements were similar to that of the fly but did not match exactly. This is due to the small gain in the motors of the mobile robot.

In this study, the fruit fly was freely moving and not tethered and can this help in long-time observation of the anima in the futurel. This research will be useful in brain imaging research and learning and interpreting the animal's behaviors. This interface can also be extended to other animals in the future.

REFERENCES

[1] Wehner, R., 2003. “Desert ant navigation: how miniature brains solve complex tasks”. *Journal of Comparative Physiology A,* **189**(8), pp. 579–588.

[2] Vo Doan, T. T., and Sato, H., 2016. “Insect-machine hybrid system: Remote radio control of a freely flying beetle (mercynorrhina torquata)”. *Journal of Visualized Experiments, JoVE*(115), p. e54260.

[3] Lal, A., Bozkurt, A., and Gilmour Jr., R. F., 2011. “Bioelectrical enhancement in tissue-electrode coupling with metamorphic-stage insertions for insect machine interfaces”. *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug, pp. 5420–5423.

[4] Son, J.-H., and Ahn, H.-S., 2014. “Bio-insect and artificial robot interaction: learning mechanism and experiment”. *Soft Computing,* **18**(6), pp. 1127–1141.

[5] Tsang, W. M., Stone, A. L., Otten, D., Aldworth, Z. N., Daniel, T. L., Hildebrand, J. G., Levine, R. B., and Voldman, J., 2012. “Insect-machine interface: A carbon nanotube-enhanced flexible neural probe”. *Journal of Neuroscience Methods,* **204**(2), pp. 355–365.

[6] Ejaz, N., Peterson, K. D., and Krapp, H. G., 2011. “An experimental platform to study the closed-loop performance of brain-machine interfaces”. *Journal of Visualized Experiments : JoVE*(49), p. 1677.

[7] Bozkurt, A., Gilmour Jr, R. F., Sinha, A., Stern, D., and Lal, A., 2009. “Insect-machine interface based neurocybernetics”. *IEEE Transactions on Biomedical Engineering,* **56**(6), pp. 1727–1733.

[8] Benvenuto, A., Sergi, F., Di Pino, G., Seidl, T., Campolo, D., Accoto, D., and Guglielmelli, E., 2009. “Beyond biomimetics: Towards insect/machine hybrid controllers for space applications”. *Advanced Robotics,* **23**(7-8), pp. 939–953.

[9] Atsushi, T., Ryo, M., Daisuke, K., and Ryohei, K., 2010. “Construction of a brain-machine hybrid system to analyze adaptive behavior of silkworm moth”. *Intelligent Robots and Systems*, pp. 2389–2394.

[10] Akimoto, K., Watanabe, S., and Yano, M., 1999. “An insect robot controlled by the emergence of gait patterns”. *Artificial Life and Robotics,* **3**(2), pp. 102–105.

[11] Tsujita, K., Tsuchiya, K., and Onat, A., 2001. “Adaptive gait pattern control of a quadruped locomotion robot”. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference*, Vol. 4, pp. 2318–2325 vol.4.

[12] Asama, H., 2007. “Mobiligence: Emergence of adaptive motor function through interaction among the body, brain and environment”. *IEEE Transactions*, March, pp. 30–31.

[13] Wessnitzer, J., Asthenidis, A., Petrou, G., and Webb, B., 2011. "A cricket-controlled robot orienting towards a sound source". In *Towards Autonomous Robotic Systems: 12th Annual Conference, TAROS 2011, Sheffield, UK, August 31 – September 2, 2011. Proceedings*, Springer Berlin Heidelberg, pp. 1–12.

[14] Melano, T., 2011. "Insect-machine interfacing". PhD thesis, The University of Arizona.

[15] Noriyasu Ando, Shuhei Emoto, R. K., 2016. "Insect-controlled Robot: A Mobile Robot Platform to Evaluate the Odor-tracking Capability of an Insect". *Journal of Visualized Experiments, JoVE*(118), Dec., p. e54802.

[16] Kain, J., Stokes, C., Gaudry, Q., Song, X., Foley, J., Wilson, R., and de Bivort, B., 2013. "Leg-tracking and automated behavioural classification in drosophila". *Nat Commun,* **4**, p. 1910.

[17] Kanzaki, R., Nagasawa, S., and Shimoyama, I., 2004. "Neural basis of odor-source searching behavior in insect microbrain systems evaluated with a mobile robot". In *Bio-mechanisms of Swimming and Flying*, N. Kato, J. Ayers, and H. Morikawa, eds., Springer Japan, pp. 155–170.

[18] Dahmen, H., Wahl, V. L., Pfeffer, S. E., Mallot, H. A., and Wittlinger, M., 2017. "Naturalistic path integration of cataglyphis desert ants on an air-cushioned lightweight spherical treadmill". *J Exp Biol,* **220**(Pt 4), pp. 634–644.

[19] Kumagai, M., and Ochiai, T., 2008. "Development of a robot balancing on a ball". In *2008 International Conference on Control, Automation and Systems*, pp. 433–438.

[20] Robinson, J. D., Holland, J. B., Hayes, M. J. D., and Langlois, R. G., 2005. "Velocity-level kinematics of the atlas spherical orienting device using omni-wheels". *Transactions of the Canadian Society for Mechanical Engineering,* **29**(4), pp. 691–700.

[21] Masatoshi, S., Hayato, K., Hajime, A., and Isao, E., 2003. "wheels for all-directional vehicle". Japan patent no. JP3421290B2.

APPENDICES

APPENDIX A
SUB-SYSTEM PROGRAM CODES

Listing 1: Image Processing Program

```
/*

Description:
  Program to compensate the movements of a Drosophila melanogaster walking
      on
  the Active Omni-directional Treadmill (AOT). This program uses OpenCV
     Library
  to perform image-processing on the images captured by a Near-Infrared
     Camera
Written by:
  Suddarsun Shivakumar and Dr. Dal Hyung Kim
  April 2018

*/

// HEADER FILES INCLUSION

#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/opencv.hpp>
#include "opencv2/core/core.hpp"
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <image_coverter/position.h>
#include <geometry_msgs/Twist.h>
#include <fstream>
#include <chrono>
#if defined(__linux__) || defined(__APPLE__)
#include <fcntl.h>
#include <termios.h>
#define STDIN_FILENO 0
#elif defined(_WIN32) || defined(_WIN64)
#include <conio.h>
#endif

#include <stdlib.h>
#include <stdio.h>
```

```
// FOR DYNAMIXEL MOTORS TO BE USED IN CONJUNCTION WITH DYNAMIXEL
   DynamixelSDK

#if defined(__linux__) || defined(__APPLE__)
#include <fcntl.h>
#include <termios.h>
#define STDIN_FILENO 0
#elif defined(_WIN32) || defined(_WIN64)
#include <conio.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include "DynamixelSDK.h"

// Control table address
#define ADDR_PRO_TORQUE_ENABLE 64
#define ADDR_PRO_GOAL_VELOCITY 104
#define ADDR_PRO_PRESENT_POSITION 132

// Data Byte Length
#define LEN_PRO_GOAL_VELOCITY 4
#define LEN_PRO_PRESENT_POSITION 4

// Protocol version
#define PROTOCOL_VERSION 2.0

// Default setting
#define DXL1_ID 100
#define DXL2_ID 101
#define DXL3_ID 102

#define BAUDRATE 1000000
#define DEVICENAME "/dev/ttyUSB0"


#define TORQUE_ENABLE 1
#define TORQUE_DISABLE 0

#define ESC_ASCII_VALUE 0x1b

int getch()
{
#if defined(__linux__) || defined(__APPLE__)
```

```
  struct termios oldt, newt;
  int ch;
  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  return ch;
#elif defined(_WIN32) || defined(_WIN64)
  return _getch();
#endif
}

int kbhit(void)
{
#if defined(__linux__) || defined(__APPLE__)
  struct termios oldt, newt;
  int ch;
  int oldf;

  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
  fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

  ch = getchar();

  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  fcntl(STDIN_FILENO, F_SETFL, oldf);

  if (ch != EOF)
  {
    ungetc(ch, stdin);
    return 1;
  }

  return 0;
#elif defined(_WIN32) || defined(_WIN64)
  return _kbhit();
#endif
}
```

```
using namespace cv;
using namespace std;
typedef std::chrono::high_resolution_clock Clock;
auto prev_clock = Clock::now();
// DECLARATION OF GLOBAL VARIABLES TO COMPUTE OMEGA 1, 2, AND 3

double theta = 0.0;
double J1M[9] = {0.0,0.000408326,-0.00856566,-0.000353621,-0.000204163,
-0.00856566,0.000353621,-0.000204163,-0.00856566}; // J inverse * M
double omega_1 = 0.0; // ANGULAR VELOCITY FOR MOTOR 1 IN AOT
double omega_2 = 0.0; // ANGULAR VELOCITY FOR MOTOR 2 IN AOT
double omega_3 = 0.0; // ANGULAR VELOCITY FOR MOTOR 3 IN AOT
double K = 100.0;
double SF = 60.0;
double prev_dth = 0.0;
double lin_vel =0.0;
double ang_vel = 0.0;
double target_location[3] = {320.0, 188.0, 0.0};
double thm = 20*3.141592/180; // IN RADIANS
double csth_thm[2] = {cos(thm), sin(thm)}; // COMPENSATION FOR CAMERA AXIS


Mat src; Mat src_gray;
int threshold_value = 125;
int const max_BINARY_value = 255;
int threshold_type = 0;
RNG rng(12345);
int repeat_time = 20;
int width = 640; // IMAGE WIDTH
int height = 376; // IMAGE HEIGHT

// FOR DYNAMIXEL MOTORS TO BE USED IN CONJUNCTION WITH DYNAMIXEL
   DynamixelSDK

dynamixel::PortHandler *portHandler = dynamixel::PortHandler::getPortHandler
   (DEVICENAME);
dynamixel::PacketHandler *packetHandler = dynamixel::PacketHandler::
   getPacketHandler(PROTOCOL_VERSION);
dynamixel::GroupSyncWrite groupSyncWrite(portHandler, packetHandler,
   ADDR_PRO_GOAL_VELOCITY, LEN_PRO_GOAL_VELOCITY);
dynamixel::GroupSyncRead groupSyncRead(portHandler, packetHandler,
   ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
 int dxl_comm_result = COMM_TX_FAIL; // Communication result
 bool dxl_addparam_result = false; // addParam result
 bool dxl_getdata_result = false; // GetParam result
```

```
 uint8_t dxl_error = 0; // Dynamixel error
 uint8_t param_goal_position[4];
 int32_t dxl1_present_position = 0, dxl2_present_position = 0,
    dxl3_present_position = 0;


// TO SAVE THE DATA
 ofstream OutFile, OutFile_Data;
 int64_t OutFile_Data_size_byte = 17*sizeof(double); // 13 numbers will be
    save per image (you may change)

// OPENING THE FILES TO WRITE DATA
void save_init(){

// my_file_name = "Fly_Image_Mar_28.bin"
  OutFile.open("Fly_Image_Mar_28.bin", ios::out | ios::binary);
  OutFile_Data.open("Fly_Data_Mar_28.bin", ios::out | ios::binary);
  //OutFile_Data.write((char *)OutFile_Data_size_byte , sizeof(
     OutFile_Data_size_byte));

}

// CLOSING THE BINARY FILES AFTER DATA IS WRITTEN

void save_deinit(){
OutFile.close();
OutFile_Data.close();

}


// INITIALIZING THE MOTORS

void motor_init(){

 if (portHandler->openPort())
{
 printf("Succeeded␣to␣open␣the␣port!\n");
}
else
{
 ROS_ERROR("Failed␣to␣open␣the␣port!\n");
 ROS_ERROR("Press␣any␣key␣to␣terminate...\n");
 getch();
}
```

```
// Set port baudrate
if (portHandler->setBaudRate(BAUDRATE))
{
 printf("Succeeded␣to␣change␣the␣baudrate!\n");
}
else
{
 ROS_ERROR("Failed␣to␣change␣the␣baudrate!\n");
 ROS_ERROR("Press␣any␣key␣to␣terminate...\n");
 getch();
}
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID,
   ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
 ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
 ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
 printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL1_ID);
}

// Enable Dynamixel#2 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID,
   ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
 ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
 ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
 printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL2_ID);
}

dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID,
   ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
```

```
if (dxl_comm_result != COMM_SUCCESS)
{
 ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
 ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
 printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL3_ID);
}

dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID, 11, 1,
    &dxl_error);
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID, 11, 1,
    &dxl_error);
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID, 11, 1,
    &dxl_error);

dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID,
   ADDR_PRO_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL1_ID);
}

// Enable Dynamixel#2 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID,
   ADDR_PRO_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
```

```
}
else
{
  printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL2_ID);
}

dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID,
    ADDR_PRO_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL3_ID);
}

dxl_addparam_result = groupSyncRead.addParam(DXL1_ID);
 if (dxl_addparam_result != true)
 {
   ROS_ERROR("[ID:%d]␣groupSyncRead␣addparam␣failed", DXL1_ID);
 }

 // Add parameter storage for Dynamixel#2 present position value
 dxl_addparam_result = groupSyncRead.addParam(DXL2_ID);
 if (dxl_addparam_result != true)
 {
   ROS_ERROR("[ID:%d]␣groupSyncRead␣addparam␣failed", DXL2_ID);
 }


 dxl_addparam_result = groupSyncRead.addParam(DXL3_ID);
 if (dxl_addparam_result != true)
 {
   ROS_ERROR("[ID:%d]␣groupSyncRead␣addparam␣failed", DXL3_ID);
 }

}


// WRITING THE VELOCITIES TO THE AOT MOTORS
```

```
void Motor_Assign(double v1, double v2, double v3)
{
  uint8_t param_vel_position[4];
  param_vel_position[0] = DXL_LOBYTE(DXL_LOWORD((int)v1));
  param_vel_position[1] = DXL_HIBYTE(DXL_LOWORD((int)v1));
  param_vel_position[2] = DXL_LOBYTE(DXL_HIWORD((int)v1));
  param_vel_position[3] = DXL_HIBYTE(DXL_HIWORD((int)v1));

dxl_comm_result = groupSyncWrite.addParam(DXL1_ID, (uint8_t*)
   param_vel_position);
 if (dxl_comm_result != true)
 ROS_ERROR("FAILED!!");


 param_vel_position[0] = DXL_LOBYTE(DXL_LOWORD((int)v2));
 param_vel_position[1] = DXL_HIBYTE(DXL_LOWORD((int)v2));
 param_vel_position[2] = DXL_LOBYTE(DXL_HIWORD((int)v2));
 param_vel_position[3] = DXL_HIBYTE(DXL_HIWORD((int)v2));

 dxl_comm_result = groupSyncWrite.addParam(DXL2_ID, (uint8_t*)
    param_vel_position);
 if (dxl_comm_result != true)
   ROS_ERROR("FAILED!!");

   param_vel_position[0] = DXL_LOBYTE(DXL_LOWORD((int)v3));
   param_vel_position[1] = DXL_HIBYTE(DXL_LOWORD((int)v3));
   param_vel_position[2] = DXL_LOBYTE(DXL_HIWORD((int)v3));
   param_vel_position[3] = DXL_HIBYTE(DXL_HIWORD((int)v3));

 dxl_comm_result = groupSyncWrite.addParam(DXL3_ID, (uint8_t*)
    param_vel_position);
 if (dxl_comm_result != true)
    ROS_ERROR("FAILED!!");

  dxl_comm_result = groupSyncWrite.txPacket();
  if (dxl_comm_result != COMM_SUCCESS) ROS_ERROR("%s\n", packetHandler->
     getTxRxResult(dxl_comm_result));

groupSyncWrite.clearParam();

}


// DEINITIALIZING THE MOTORS
```

```
void motor_deinit()
{
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID,
     ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
   ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
  }
  else if (dxl_error != 0)
  {
   ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
  }
  else
  {
   printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL1_ID);
  }

  // Enable Dynamixel#2 Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID,
     ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
   ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
  }
  else if (dxl_error != 0)
  {
   ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
  }
  else
  {
   printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL2_ID);
  }

  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID,
     ADDR_PRO_TORQUE_ENABLE, 0, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
   ROS_ERROR("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
  }
  else if (dxl_error != 0)
  {
   ROS_ERROR("%s\n", packetHandler->getRxPacketError(dxl_error));
  }
  else
```

```
  {
   printf("Dynamixel#%d␣has␣been␣successfully␣connected␣\n", DXL3_ID);
  }


}

static const std::string OPENCV_WINDOW = "Image␣window";


class ImageConverter
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Subscriber image_sub_;
  image_transport::Publisher image_pub_;
  ros::Publisher Fly_Pos;
  ros::Subscriber seq_sub;

public:
  ImageConverter()
    : it_(nh_)
  {
    // Subscrive to input video feed and publish output video feed
    ros::NodeHandle nh;
    Fly_Pos = nh.advertise<geometry_msgs::Twist>("/cmd_vel", 1000); //
        PUBLISHING INPUT FOR THE MOBILE ROBOT

    image_sub_ = it_.subscribe("/image_raw", 1,
      &ImageConverter::imageCb, this); // SUBSCRIBING TO THE RAW IMAGES
          FROM CAMERA


    image_pub_ = it_.advertise("/image_converter/output_video", 1);


    cv::namedWindow(OPENCV_WINDOW);
  }

~ImageConverter()
  {
  cv::destroyWindow(OPENCV_WINDOW);
 }

  void imageCb(const sensor_msgs::ImageConstPtr& msg)
```

```
{

  int sequence_now=0;
  double seq_init;
  geometry_msgs::Twist MSG; //INSTANCE TO PUBLISH MESSAGE
  cv_bridge::CvImagePtr cv_ptr;

  std_msgs::Header h = msg->header; //TO GET FRAME NUMBER


  // time start

  try
  {
    cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::MONO8)
       ;
    seq_init=h.seq;
    sequence_now=h.seq%(repeat_time*3);

  }
  catch (cv_bridge::Exception& e)
  {
    ROS_ERROR("cv_bridge␣exception:␣%s", e.what());
    return;
  }


  auto begin = Clock::now();

  auto samplingtime_secs = std::chrono::duration_cast<std::chrono::
     microseconds>(begin - prev_clock).count() ; //LOOP TIME
  prev_clock = begin;

  Mat im = cv_ptr->image;
  Mat im_bw;

  // IMAGE PROCESSING

  threshold(im, im_bw, 50.0, 255.0, 0);
  vector<vector<Point> > contours;
  vector<Vec4i> hierarchy;
  int savedContour = -1; // contour index (largest contour)
  double maxArea = 0.0; // size of the largest contour
  findContours(im_bw, contours, hierarchy, CV_RETR_EXTERNAL,
     CV_CHAIN_APPROX_SIMPLE, Point(0,0));
```

```
for (int i = 0; i< contours.size(); i++)
{
    double area = contourArea(contours[i]);
    if (area > maxArea)
    {
        maxArea = area;
        savedContour = i;
    }

}

// COMPTUTING THE CENTROID USING THE MOMENTS

double _cx = target_location[0]; double _cy = target_location[1]; double
     _theta = target_location[2];
if (maxArea > 100.0) {
  Moments mu = moments(contours[savedContour], false );
  // compute centroid & orientation*0.01
  _cx = mu.m10/mu.m00;
  _cy = mu.m01/mu.m00;
  double mu20p = mu.m20/mu.m00-_cx*_cx;
  double mu02p= mu.m02/mu.m00-_cy*_cy;
  double mu11p= mu.m11/mu.m00-_cx*_cy;
  _theta = atan2(2*mu11p,(mu20p-mu02p))/2; // in radian
}

// COMPUTING THE DEL. X, DEL. Y AND DEL. THETA NEEDED TO CALCULATE THE
// MOTOR VELOCITIES

double dx = (_cx-target_location[0]);
double dy = (_cy-target_location[1]);
double dth = -(_theta - target_location[2]);

// ACCOUNTNIG FOR THE DIFFERNECE IN ALLIGNMENT OF CAMERA AND MOTOR AXES

double dxm = csth_thm[0]*dx - csth_thm[1]*dy;
double dym = csth_thm[1]*dx + csth_thm[0]*dy;
double dthm = dth ;//+ thm;

// SETTING TOLERANCE

double limit_r = 50;
double limit_th = 3*3.141592/180;

// COMPUTING AOT MOTOR VELOCITIES
```

```
  if ((sqrt(dx*dx + dy*dy) > limit_r) || (fabs(dth) > limit_th)) {

    omega_1 = (J1M[0]*dxm + J1M[1]*dym + J1M[2]*dthm*SF)*K;
      omega_2 = (J1M[3]*dxm + J1M[4]*dym + J1M[5]*dthm*SF)*K;
      omega_3 = (J1M[6]*dxm + J1M[7]*dym + J1M[8]*dthm*SF)*K;
  }
  else {
    omega_1 = 0; omega_2 =0; omega_3 = 0;
  }

  // CALCULATING THE LINEAR AND ANGULAR VELOCITIES FOR THE ROBOT

  if ((sqrt(dx*dx + dy*dy) > limit_r)) {
    lin_vel = (sqrt(dx*dx + dy*dy)*38.40983291722681*0.01)/(
        samplingtime_secs*0.001);
  }

  else lin_vel =0;

  if((fabs(dth) > limit_th))
    ang_vel = (dthm/(samplingtime_secs*0.001))*57;

  else ang_vel = 0;

// PUBLISHING THE LINEAR AND ANGULAR VELOCITIES FOR THE MOBILE ROBOT

 MSG.linear.x= lin_vel;
 MSG.angular.z= ang_vel;

// DRAWING THE CONTOUR CENTROID ON THE IMAGE
  Mat drawing = Mat::zeros( im_bw.size(), CV_8UC3 );
  cv::cvtColor(im, drawing, cv::COLOR_GRAY2BGR);
  //im.copyTo(drawing);
  if (maxArea > 0.0) {
    Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.
        uniform(0,255) );
    drawContours(drawing, contours, savedContour, color, 2, 8, hierarchy,
        0, Point() );
  }
  circle( drawing, Point(_cx,_cy), 3, Scalar(255,0,255), 1, 8, 0);
  circle( drawing, Point(target_location[0],target_location[1]), limit_r,
     Scalar(0,0,255), 1, 8, 0);
  imshow("a",drawing);
  waitKey(1);
```

```
// WRITING VELOCITIES TO MOTORS

  Motor_Assign(omega_1,omega_2,omega_3);


// READING CURRENT POSITION OF THE MOTOR AFTER COMPENSATION

   dxl_comm_result = groupSyncRead.txRxPacket();
   if (dxl_comm_result != COMM_SUCCESS) ROS_ERROR("%s\n", packetHandler->
      getTxRxResult(dxl_comm_result));

   // Check if groupsyncread data of Dynamixel#1 is available
   dxl_getdata_result = groupSyncRead.isAvailable(DXL1_ID,
      ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
   if (dxl_getdata_result != true)
   {
     ROS_ERROR( "[ID:%d] groupSyncRead getdata failed", DXL1_ID);
   }

   // Check if groupsyncread data of Dynamixel#2 is available
   dxl_getdata_result = groupSyncRead.isAvailable(DXL2_ID,
      ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
   if (dxl_getdata_result != true)
   {
     ROS_ERROR( "[ID:%d] groupSyncRead getdata failed", DXL2_ID);
   }

   dxl_getdata_result = groupSyncRead.isAvailable(DXL3_ID,
      ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
   if (dxl_getdata_result != true)
   {
     ROS_ERROR( "[ID:%d] groupSyncRead getdata failed", DXL3_ID);
   }

   // Get Dynamixel#1 present position value
   dxl1_present_position = groupSyncRead.getData(DXL1_ID,
      ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);

   // Get Dynamixel#2 present position value
   dxl2_present_position = groupSyncRead.getData(DXL2_ID,
      ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
```

```
     dxl3_present_position = groupSyncRead.getData(DXL3_ID,
        ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);



    auto end = Clock::now();
    double elapsed_secs = std::chrono::duration_cast<std::chrono::
       microseconds>(end - begin).count() ;
    ROS_INFO ("%0.3f,%0.3f", (double)elapsed_secs/1000,(double)
       samplingtime_secs/1000);




    // Output modified video stream
    image_pub_.publish(cv_ptr->toImageMsg());
    Fly_Pos.publish(MSG);

    //Writing Image AND data to Binary File
    OutFile.write((char *)im.data, width*height*sizeof(uchar));
    double secs = ros::Time::now().toSec();
    double output_data[] = {seq_init, elapsed_secs, (double)
       samplingtime_secs, _cx, _cy, _theta, dxm, dym, dthm,
    omega_1, omega_2, omega_3, maxArea, (double)dxl1_present_position, (
       double)dxl2_present_position,
    (double)dxl3_present_position, secs}; // 17 variables
    OutFile_Data.write((char *)output_data, OutFile_Data_size_byte);


  }
};

int main(int argc, char** argv)
{

  // INITIALIZING ROS NODE
  ros::init(argc, argv, "image_converter");

  save_init(); // OPENING FILES
  motor_init(); // INITIALIZING MOTORS
  ImageConverter ic;
  ros::spin();

  motor_deinit(); // DEINITIALIZING MOTORS
  save_deinit(); // CLOSING FILES
```

```
  return 0;
}
```

Listing 2: Code to obtain position data from the mobile robot

```
/*

Description:
  Program to obtain the encoder data from the right and left wheel of the
     mobile
  robot. The position obtained is published by the robot to the
     sensor_state topic
Written by:
  Suddarsun Shivakumar and Dr. Dal Hyung Kim
  April 2018

*/


#include "ros/ros.h"
#include <turtlebot3_msgs/SensorState.h>
#include <fstream>

using namespace std;

ofstream Output; // Declaring the file handler
int64_t Output_size_byte = 3*sizeof(int64_t);

// Callback function called everytime a message is published to the
   subsribed topic

void CallBack(const turtlebot3_msgs::SensorState::ConstPtr& data){
  int64_t usecs = (int64_t)(ros::Time::now().toSec()*1000000);
  int64_t output_data[] = {usecs, data->left_encoder,data->right_encoder};
     // Variables in the file: time, left encoder data, right encoder data
  Output.write((char *)output_data, Output_size_byte); // Writing the data
     obtianed to a binary file
}

int main(int argc, char** argv)

{
  Output.open("MobRob_Data_9:54PM_Mar29_2018.bin", ios::out | ios::binary);
     // Opening the binary file
  ros::init(argc, argv, "mobrob_data_tracker"); // initializing node
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("sensor_state", 1000, CallBack); //
```

```
    subscribing to "sensor_state" topic
  ros::spin();
  Output.close(); // Closing the binary file after writing the data

  return 0;

}
```

Listing 3: Launch file that launches all nodes on the host system

```
<launch>
  <!-- This launch file is to launch all nodes in one command from the
      terminal
      of the host system. This file launches the following nodes:
      The image procesing node
      The mobile robot tracker node
      The camera node to capture images
      It parameters of the camera are also configured in this file prior to
          launch
    -->
  <machine name="local_alt" address="localhost" default="true"/>

  <!-- Launch Nodes -->
  <node name="image_converter" pkg="image_coverter" type="
     image_coverter_node"/>
  <node name="mobrob_data_tracker" pkg="mobrob_data_tracker" type="
     mobrob_data_tracker_node"/>
  <node name="pointgrey_camera_node" pkg="pointgrey_camera_driver" type="
     camera_node"/>

  <!-- Configuring camera parameters -->

  <!-- Setting the size of the image -->
  <param name="/pointgrey_camera_node/format7_roi_height" value= "376"/>
  <param name="/pointgrey_camera_node/format7_roi_width" value= "640"/>
  <param name="/pointgrey_camera_node/format7_x_offset" value= "732"/>
  <param name="/pointgrey_camera_node/format7_y_offset" value= "596"/>

  <!-- Adjusting the exposure settings -->
  <param name="/pointgrey_camera_node/auto_exposure" value= "false"/>
  <param name="/pointgrey_camera_node/exposure" value= "-7.585"/>
  <param name="/pointgrey_camera_node/auto_sharpness" value= "false"/>
  <param name="/pointgrey_camera_node/sharpness" value= "1525"/>
  <param name="/pointgrey_camera_node/auto_shutter" value= "false"/>
  <param name="/pointgrey_camera_node/shutter" value= "1"/>
  <param name="/pointgrey_camera_node/auto_gain" value= "false"/>
  <param name="/pointgrey_camera_node/gain" value= "20"/>
  <param name="/pointgrey_camera_node/frame_rate" value= "66"/>

  <!-- Setting the output to enable LED pulsing -->
  <param name="/pointgrey_camera_node/enable_strobe2" value= "true"/>
  <param name="/pointgrey_camera_node/strobe2_polarity" value= "1"/>
```

```
  <param name="/pointgrey_camera_node/strobe2_duration" value= "0.5"/>

</launch>
```

Listing 4: Program to move the mobile robot

```
// NOTE: ONLY RELEVANT PARTS OF THE CODE ARE LISTED HERE
// This code was written using Arduino IDE

#include "turtlebot3_core_config.h"
/*******************************************************
* ROS NodeHandle
*******************************************************/
ros::NodeHandle nh;

/*******************************************************
* Subscriber
*******************************************************/
ros::Subscriber<geometry_msgs::Twist> cmd_vel_sub("cmd_vel",
   commandVelocityCallback);

/*******************************************************
* Publisher
*******************************************************/
turtlebot3_msgs::SensorState sensor_state_msg;
ros::Publisher sensor_state_pub("sensor_state", &sensor_state_msg);
double goal_linear_velocity = 0.0;
double goal_angular_velocity = 0.0;

/*******************************************************
* Setup function
*******************************************************/
void setup()
{
  // Initialize ROS node handle, advertise and subscribe the topics
  nh.initNode();
  nh.getHardware()->setBaud(57600);
  nh.subscribe(cmd_vel_sub);
  nh.advertise(sensor_state_pub);

  pinMode(13, OUTPUT);

  SerialBT2.begin(57600);
  prev_update_time = millis();
  setup_end = true;
}

/*******************************************************
```

```
* SoftwareTimer of Turtlebot3
*******************************************************************/
static uint32_t tTime[4];

/*********************************************************
* Loop function
*********************************************************/
void loop()
{
  receiveRemoteControlData();

  if ((millis()-tTime[0]) >= (1000 / CONTROL_MOTOR_SPEED_PERIOD))
  {
   // controlMotorSpeed();
              AOTMotorControl();
    tTime[0] = millis();
  }

    nh.spin();

}


/*********************************************************
Callback function for cmd_vel msg
*********************************************************/
void commandVelocityCallback(const geometry_msgs::Twist& cmd_vel_msg)
{
  goal_linear_velocity = cmd_vel_msg.linear.x;
  goal_angular_velocity = cmd_vel_msg.angular.z;

}

/*********************************************************
* Publish msgs (sensor_state: encoders)
*********************************************************/
void publishSensorStateMsg(void)
{
  bool dxl_comm_result = false;

  int32_t current_tick;

  sensor_state_msg.stamp = nh.now();
  sensor_state_msg.battery = checkVoltage();
```

```
  dxl_comm_result = motor_driver.readEncoder(sensor_state_msg.left_encoder,
     sensor_state_msg.right_encoder);

  if (dxl_comm_result == true)
  {
    sensor_state_pub.publish(&sensor_state_msg);
  }
  else
  {
    return;
  }

}

/*************************************************************
* Control mobile robot velocity
*************************************************************/
void AOTMotorControl(void)
{
       bool dxl_comm_result = false;
       double omega_1 = 0.0;
       double omega_2 = 0.0;
       double omega_3 = 0.0;

  double wheel_speed_cmd[2];
  double lin_vel1;
  double lin_vel2;

  wheel_speed_cmd[LEFT] = goal_linear_velocity - (goal_angular_velocity *
     WHEEL_SEPARATION / 2);
  wheel_speed_cmd[RIGHT] = goal_linear_velocity + (goal_angular_velocity *
     WHEEL_SEPARATION / 2);

  lin_vel1 = wheel_speed_cmd[LEFT] * VELOCITY_CONSTANT_VALUE;
  if (lin_vel1 > LIMIT_X_MAX_VELOCITY)
  {
    lin_vel1 = LIMIT_X_MAX_VELOCITY;
  }
  else if (lin_vel1 < -LIMIT_X_MAX_VELOCITY)
  {
    lin_vel1 = -LIMIT_X_MAX_VELOCITY;
  }

  lin_vel2 = wheel_speed_cmd[RIGHT] * VELOCITY_CONSTANT_VALUE;
  if (lin_vel2 > LIMIT_X_MAX_VELOCITY)
```

```
  {
    lin_vel2 = LIMIT_X_MAX_VELOCITY;
  }
  else if (lin_vel2 < -LIMIT_X_MAX_VELOCITY)
  {
    lin_vel2 = -LIMIT_X_MAX_VELOCITY;
  }

 dxl_comm_result = motor_driver.speedControl_AOT((int64_t)omega_1, (int64_t)
     omega_2, (int64_t)omega_3, (int64_t)lin_vel1, (int64_t)lin_vel2);

 if (dxl_comm_result == false)
           return;

}

// Excerpt from turtlebot3_motor_driver.cpp

#include "turtlebot3_motor_driver.h"


Turtlebot3MotorDriver::Turtlebot3MotorDriver()
: baudrate_(BAUDRATE),
  protocol_version_(PROTOCOL_VERSION),
  left_wheel_id_(DXL_LEFT_ID),
  right_wheel_id_(DXL_RIGHT_ID),
 {
}

Turtlebot3MotorDriver::~Turtlebot3MotorDriver()
{
  closeDynamixel();
}

bool Turtlebot3MotorDriver::init(void)
{
  portHandler_ = dynamixel::PortHandler::getPortHandler(DEVICENAME);
  packetHandler_ = dynamixel::PacketHandler::getPacketHandler(
      PROTOCOL_VERSION);

  // Open port
  if (portHandler_->openPort())
  {
    #ifdef DEBUG
    sprintf(log_msg, "Port␣is␣Opened");
```

```
    nh.loginfo(log_msg);
    #endif
  }
  else
  {
    return false;
  }

  // Set port baudrate
  if (portHandler_->setBaudRate(baudrate_))
  {
    #ifdef DEBUG
    sprintf(log_msg, "Baudrate␣is␣set");
    nh.loginfo(log_msg);
    #endif
  }
  else
  {
    return false;
  }

  // Enable Dynamixel Torque
  setTorque(left_wheel_id_, true);
  setTorque(right_wheel_id_, true);
  return true;
}

bool Turtlebot3MotorDriver::setTorque(uint8_t id, bool onoff)
{
  uint8_t dxl_error = 0;
  int dxl_comm_result = COMM_TX_FAIL;
  dxl_comm_result = packetHandler_->write1ByteTxRx(portHandler_, id, 11, 1,
     &dxl_error);
  dxl_comm_result = packetHandler_->write1ByteTxRx(portHandler_, id,
     ADDR_X_TORQUE_ENABLE, onoff, &dxl_error);

  if(dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler_->printTxRxResult(dxl_comm_result);
  }
  else if(dxl_error != 0)
  {
    packetHandler_->printRxPacketError(dxl_error);
  }
}
```

```
void Turtlebot3MotorDriver::closeDynamixel(void)
{
  // Disable Dynamixel Torque
  setTorque(left_wheel_id_, false);
  setTorque(right_wheel_id_, false);

  // Close port
  portHandler_->closePort();
}

bool Turtlebot3MotorDriver::readEncoder(int32_t &left_value, int32_t &
    right_value)
{
  int dxl_comm_result = COMM_TX_FAIL; // Communication result
  bool dxl_addparam_result = false; // addParam result
  bool dxl_getdata_result = false; // GetParam result

  // Set parameter
  dxl_addparam_result = groupSyncReadEncoder_->addParam(left_wheel_id_);
  if (dxl_addparam_result != true)
    return false;

  dxl_addparam_result = groupSyncReadEncoder_->addParam(right_wheel_id_);
  if (dxl_addparam_result != true)
    return false;

  // Syncread present position
  dxl_comm_result = groupSyncReadEncoder_->txRxPacket();
  if (dxl_comm_result != COMM_SUCCESS)
    packetHandler_->printTxRxResult(dxl_comm_result);

  // Check if groupSyncRead data of Dynamixels are available
  dxl_getdata_result = groupSyncReadEncoder_->isAvailable(left_wheel_id_,
      ADDR_X_PRESENT_POSITION, LEN_X_PRESENT_POSITION);
  if (dxl_getdata_result != true)
    return false;

  dxl_getdata_result = groupSyncReadEncoder_->isAvailable(right_wheel_id_,
      ADDR_X_PRESENT_POSITION, LEN_X_PRESENT_POSITION);
  if (dxl_getdata_result != true)
    return false;

  // Get data
  left_value = groupSyncReadEncoder_->getData(left_wheel_id_,
```

```
    ADDR_X_PRESENT_POSITION, LEN_X_PRESENT_POSITION);
  right_value = groupSyncReadEncoder_->getData(right_wheel_id_,
    ADDR_X_PRESENT_POSITION, LEN_X_PRESENT_POSITION);

  groupSyncReadEncoder_->clearParam();
  return true;
}


bool Turtlebot3MotorDriver::speedControl_AOT(int64_t omega_1, int64_t
   omega_2, int64_t omega_3,
int64_t left_wheel_value, int64_t right_wheel_value)
{
  bool dxl_addparam_result_;
  uint8_t dxl_comm_result_;

 dxl_addparam_result_ = groupSyncWriteVelocity_->addParam(left_wheel_id_, (
    uint8_t*)&left_wheel_value);
 if (dxl_addparam_result_ != true)
  return false;

  dxl_addparam_result_ = groupSyncWriteVelocity_->addParam(right_wheel_id_,
    (uint8_t*)&right_wheel_value);
  if (dxl_addparam_result_ != true)
  return false;

  dxl_comm_result_ = groupSyncWriteVelocity_->txPacket();
  if (dxl_comm_result_ != COMM_SUCCESS)
  {
    packetHandler_->printTxRxResult(dxl_comm_result_);
    return false;
  }

  groupSyncWriteVelocity_->clearParam();
  return true;
}
```

VITA

Graduate School
Southern Illinois University

Suddarsun Shivakumar
suddarsun@gmail.com
Southern Illinois University Carbondale
Bachelor of Science, Mechanical Engineering, May 2016
Thesis Title:
Machine-Insect Interface: Spatial Navigation of a Mobile Robot by a *Drosophila*
Major Professor: Dr. Dal Hyung Kim