

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

REMOTE CONTROL FUNCTIONALITY FOR THE
AUTONOMOUS AMIGA TRACTOR

A senior thesis submitted in partial satisfaction of the
requirements for the degree of

BACHELOR OF SCIENCE

in


ROBOTICS ENGINEERING

by

Samuel JM Leveau

June 2025

The Senior Thesis of Samuel JM Leveau
is approved:

DocuSigned by:

5F739D6B0881448...
Professor Dejan Milutinovic, Advisor

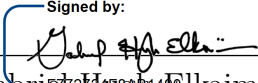
Signed by:

72F493A8149C...
Professor Gabriel H. Elkaim

Table of Contents

List of Figures	iv
List of Tables	v
Abstract	vi
Dedication	vii
Acknowledgments	viii
1 Introduction	1
1.1 Thesis Overview	3
1.2 Deliverables	4
1.3 Contributions	4
1.4 Current Research	6
1.5 Thesis Outline	7
2 Amiga Platform	9
2.1 Kinematics	12
2.2 Software	14
3 Communication	17
3.1 Server	20
3.2 Client	22
3.3 Message Structure	24
4 Kivy	25
4.1 Custom Widgets	29
4.2 Virtual Joystick	31
5 Mobile Application	34
5.1 The <i>Amigas</i> Main Process	34
5.2 Graphical User Interface	35

- 5.2.1 SETUP Tab 35
 - 5.2.2 CONTROL Tab 37
 - 5.3 Typical Workflow 39
 - 5.4 Results 40
 - 5.4.1 Responsiveness 40
 - 5.4.2 Functionality 41
- 6 Conclusion 43**
 - 6.1 Future Improvement 44
 - 6.2 Personal Reflection 44
- Bibliography 46**

List of Figures

2.1	Base Amiga [2]	10
2.2	CAN bus Network [2]	10
2.3	Emergency Shutoff Circuit [2]	11
2.4	4×4 skid-steer <i>Amiga</i> kinematic schematic: $\vec{i}, \vec{j}, \vec{k}$ denotes the global inertial frame unit vectors; L, R are the combined left and right wheel points; $\vec{V}_L, \vec{V}_C, \vec{V}_R$ are the velocity vectors at the left wheel, vehicles center C , and right wheel respectively; W_L/W_R indicate the wheel angular speeds; the dashed line from L to R represents the distance D ; heading angle is represented as θ ; and $\omega = \dot{\theta}$ as the angular velocity of the robot about the \hat{k} axis (out of the plane).	12
3.1	Server/Client Communication Block Diagram Structure	19
4.1	Kivy Graphical Architecture [7]	26
4.2	Hello World Example	27
4.3	Custom Toggle Button in its two States	30
4.4	Joystick Widget Model	31
4.5	Virtual Joystick	33
5.1	SETUP Tabs	36
5.2	CONTROL Tabs	37
5.3	Functionality Trial Run Configuration	42

List of Tables

5.1	Measured round-trip latencies for ten trials.	41
5.2	Obstacle course completion times across control modes.	42

Abstract

Remote Control Functionality for the Autonomous Amiga Tractor

by

Samuel JM Leveau

This thesis aims to address the challenges facing California's organic agriculture industry - labor shortages, hazardous working conditions, and soil degradation - by extending the functionalities of the *Amiga*, a small modular electric tractor. A mobile application was created using Kivy, Buildozer, and a User-Datagram-Protocol communication structure. The app provides a virtual joystick, live video-feed, and state driven control for autonomous navigation, emergency stop, and reverse-in-replay. This project addresses the need for human-robot interaction while demonstrating the importance of user-centered design in the agricultural setting.

To my family,

who provide unconditional love and patience towards someone who is still trying
to figure out the world for themselves.

Acknowledgments

First and foremost, I would like to thank my friends and family, who have always continued to support and push me towards excellence. Without their love and motivation, I would not have reached this point in my academic and personal growth.

I would also like to thank my peers and mentors within the Robotics and Control Research Lab at UC Santa Cruz. It has continued to be an open-door for curiosity, discussion, collaboration, and perseverance both in and out of the engineering mindset. I am especially grateful to my advisor, Dejan Milutinović, for his insightful and invaluable guidance throughout my time as an undergraduate student.

Thank you all.

Chapter 1

Introduction

With over a third of the country's vegetables, fruits, and nuts grown in the state, California stands out as a significant agricultural hub within the United States [1]. However, the cultivation and harvesting of crops in California and around the world face challenges, including labor shortages, inefficient operation, environmental concern, and labor-related hazards.

Within California, the agricultural industry has been facing heavy labor shortages over the past decade. According to a UC Davis study conducted in May 2019, Professor J. Edward Taylor completed a long-term study showing that fewer people are interested in picking crops. Results from the survey described that 56% of the respondents were not able to hire the number of workers they desired. In an attempt to mitigate this issue, 86% raised wages, and 31% reported switching acreage to less labor-intensive crops. However, these attempts were ineffective in attracting new workers [11].

This outcome may be partly due to the inherent dangers of farm labor. California cash crops such as grapes, almonds, pistachios, and strawberries rely on conventional agricultural practices that expose workers to long hours of bending, lifting, repetitive motions, and exposure to pesticides/herbicides [12]. Additionally, large machinery that assists in cultivation practices is a risk to its operators. Tractor rollovers alone account for more than half of farm-related deaths and cause 7 in 10 farms to go out of business within the year of accident [4].

Beyond worker safety, traditional agricultural practices reduce the longevity of soil. The continuous use of pesticides and herbicides reduces the microbial biodiversity and can suppress enzyme activities, compromising the soil structure and its nutrients [10]. Once fully weakened, the soil is incapable of supporting healthy plant growth, requiring more chemical products and continuing a perpetual harmful cycle.

Organic farming methods attempt to eliminate the use of agrochemicals, yet this practice still remains labor-intensive, if not more so. A recent study conducted on organic farms in New Mexico highlighted the physical demands of work, particularly the prolonged hours and the range of tasks contributing to physical stress among workers. Sprains, strains, and tears were the most prevalent injuries. There is an urgent need for intervention against such risks [3].

1.1 Thesis Overview

To address the growing need of safer and more sustainable agricultural tools, this senior thesis project leverages a small modular electric tractor: The *Amiga*. Developed by *Farm_NG* [2], the *Amiga* allows farmers to attach mechanical peripherals, such as finger weeders, to weed crops without the use of harmful pesticide and herbicides. Its compact size and fully electric system also reduce emissions and cost, offering a more affordable product to assist in labor shortages in growing organic farms. However, the tractor still requires workers to be in close physical proximity for emergency interventions and basic use. This leaves a vital gap in human-machine interaction.

This senior thesis project introduces a mobile application that creates remote control functionality for the *Amiga*, allowing the user to operate the tractor from a smartphone or tablet. In 2023, an autonomous plant-line following navigation algorithm was developed and implemented onto the *Amiga* by the Robotics and Control Lab at UC Santa Cruz. Now, the mobile application introduces human-robot interaction. Ideally, a farmer would approach the parked *Amiga*, power it on, and use the virtual joystick to wirelessly navigate the tractor to the plant-line before starting autonomous navigation. If something were to go wrong during operation down the plant-line, the user should be able to stop the *Amiga*, reverse its recent actions, make necessary adjustments, and continue navigation.

1.2 Deliverables

This thesis work conducted at the University of California Santa Cruz, was part of an ongoing collaboration between the Robotics and Control Lab and the Center for Agroecology, focused on the development and application of the *Amiga* tractor. After discussion with both collaborators and under the supervision of my principal investigator Dejan Milutinović, the senior thesis deliverables included:

- Mobile Application: mobile application that allows multiple clients, i.e. operators, to remotely control the tractor from any location on the farm. Features include:
 - Live video streaming: live streams the *Amiga* camera feed
 - Virtual Joystick: control the *Amigas* velocity and turning rate
 - Automation toggling: enable and disable previously developed autonomous navigation
 - Replay-in-Reverse: reverse the autonomous movements that just occurred
 - Emergency stop: stop the *Amiga* during any operation

1.3 Contributions

By introducing a mobile application, this project aims to transform and impact how workers interact with their equipment, shifting from physically demanding and high risk labor to a more intuitive operator role. Real-time video streaming allows the operator to monitor the *Amiga* navigation status from a much greater distance,

reducing the physical presence in the field. The joystick feature further alleviates the need for physical proximity, providing control without being near the tractor. Moreover, the combination of the joystick and camera feed alone allows the operator to navigate the *Amiga* around the field without having to view the *Amiga* at all. Similarly, the navigation toggle eliminates the need for manually starting autonomous navigation.

The replay-in-reverse functionality impacts both the operator and developer. To the operator, if the autonomous navigation fails, this feature repositions the tractor to a prior pose¹. This eliminates the need for manual correction from a close distance and reduces downtime. For developers, this method of movement could be used in later research applications. For instance, coupled with a non-linear controller, the *Amiga* could define navigation paths for self-parking or other autonomous movements.

One of the unique aspects of the mobile application is its customization, aligning with the right-to-repair movement. Unlike other systems, if the application became open-source, it would allow farmers to modify it to their specific needs. Additionally, the application is designed with versatility in mind. Utilizing a User-Datagram-Protocol (UDP) server/client internet connection, the app can connect to any other device that requires a joystick interface, not just the *Amiga* tractor. This cross-compatibility makes it a valuable tool for both farmers and developers alike.

Together, these capabilities aim to impact the agricultural sector to be safer, more intuitive, and a less arduous method of farming. Additionally, the modular design of the application aligns with the right to repair movement, allowing developers, users,

¹Position and orientation of an object in space.

and farmers to adapt, fix, or extend the tool.

1.4 Current Research

Large scale agricultural machinery companies such as AVL Motion, Kubota Corporation, and Monarch Tractors all sell functionally autonomous tractors. One of which is fully electric. In turn, these products often come with a large price tag and proprietary systems. Once bought, farmers are typically locked into using specific software and hardware configurations, limiting their ability to modify the equipment and reducing accessibility. For example, John Deere, a popular large scale farming company, requires the use of its own interactive software tool called *Service ADVISOR* which is only available to authorized dealers [5]. This prevents users from repairing their equipment themselves.

In contrast, the *Amiga* and mobile application aim to provide a more affordable and open source application for organic farmers. In particular, the *Amigas'* open-source software development kit (SDK) empowers users to optimize its system to their preferences. Similarly, the mobile application is meant to be open source, modular, and intuitive to use.

While mobile application joysticks for robots, i.e. a wireless connection between a mobile device and robot, are common in healthcare, consumer, and industrial sectors, there are only few direct virtual joystick applications for the agricultural sector. Applications such as XTEND, AkerScout, and Connect-Mobile are commonly used with

farm equipment, however, do not include a virtual joystick to manipulate its pose [6].

Developers at *Farm.ng* have created and implemented a virtual joystick module onto its on-board dashboard [2]. However, this application requires close physical proximity and does not support remote operation. While it demonstrates basic joystick control, a good baseline for the deliverable outlined in this senior thesis, it does not support video streaming, mode switching, or any other functionality.

The UDP communication protocol is used in applications that require low latency, such as online gaming, video streaming, and specific medical devices. However, its implementation in agricultural robotics, specifically with a mobile application, is relatively novel. The software development within applications such as XTEND, Aker-Scout, and Connect-Mobile, do not specify their communication protocols. Most likely, these applications rely on communication methods like Bluetooth or machine to machine communication. These systems offer stable and efficient connections over varying distances, however, UDP provides an advantage in scenarios where speed is paramount. Additionally, UDP supports multi-casting, allowing the transmission of data to multiple devices simultaneously. To multiple operators, this could be beneficial in maneuvering a tractor across a field in large distances.

1.5 Thesis Outline

The outline of this senior thesis is as follows:

Chapter 2 introduces the Base Amiga, Intelligence Kit and previously devel-

oped software central to the *Amiga*. Additionally, the *Amigas* kinematics are modeled for use in later Chapters.

Chapter 3 outlines the wireless server-client architecture between mobile application and the *Amiga*. The chapter details the communication protocol and how it navigates around limitations using asynchronous and threading tasks.

Chapter 4 provides an overview of *Kivy*, the primary graphical-user-interface (GUI) framework used in this senior thesis. Elements such as custom toggle buttons and a virtual Joystick are explained in detail and later used in Chapter 5.

Chapter 5 finally outlines the complete development and evaluation of the mobile application. It describes how the graphical interface and communication protocols described in Chapter 3 and 4 are integrated together. Additionally, performance evaluations of reactivity and functionality are tested and discussed.

Chapter 2

Amiga Platform

The *Amiga* is a small modular electric tractor developed by *Farm_NG* for farmers, educators, and developers to automate tasks, reduce carbon footprint, and minimize general maintenance. Based in *Watsonville* California, the developer team at *Farm_NG* pride themselves on collaborating with organic farmers for feedback and share the belief that food security is national security [2].

At the UC Santa Cruz farm, the *Amiga* is co-owned by the Robotics and Control Lab and the Center for Agroecology. The robotic system comprises two primary components: the *Base Amiga* and *Intelligence Kit*.

The *Base Amiga* refers to the mechanical modular platform (Figure 2.1). Without large physical peripherals, such as a seat, the tractors framework weighs approximately 250 lb. It employs a 4x4 skid-steer drive train, powered by four 500-watt motors connected to two dual Li-ion batteries. As depicted in Figure 2.2, each motor is independently controlled by its own motor controller and connected to a shared CAN bus



Figure 2.1: Base Amiga [2]

network. This can be accessed and monitored from the on-board dashboard and pendant (a physical joystick). In addition to the CAN bus network, the system includes an

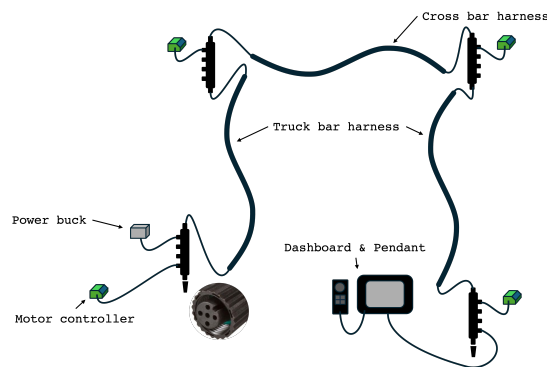


Figure 2.2: CAN bus Network [2]

emergency stop circuit (Figure 2.3) that cuts power to each individual motor controller. This method stops the *Amiga* immediately but does not engage a physical brake, which may allow it to roll freely, a potential safety hazard.

Intelligence Kit: To enable advanced control and autonomy over the *Amiga*, the intelligence kit includes a Brain, which encompasses the *Amiga* Software Develop-

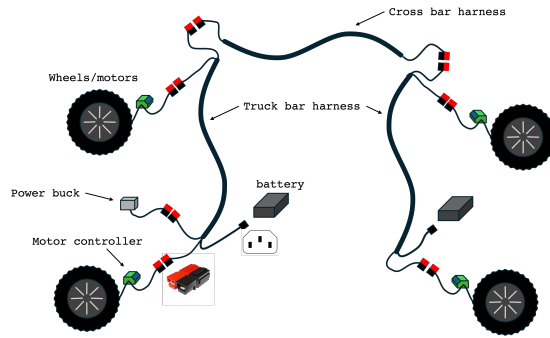


Figure 2.3: Emergency Shutoff Circuit [2]

ment Kit (SDK) on a Xavier NX processing unit. This kit also includes a RTK GPS, two Luxonis OAK-D cameras, and a Power over Ethernet switch (PoE). The Brain is responsible for running applications that interact with motors and other software services. The Brain is connected to the *CAN* bus network (from the Base *Amiga*) and OAK cameras through the PoE switch. To switch between pendant and brain control, the AUTO CONTROL button can be toggled on the Dashboard. Note: the brain draws power from the *CAN* bus connector, while the cameras, OAK1 and OAK2, use PoE. The Brain also provides wireless connectivity via WiFi and an external antenna. This wireless connection allows developers to connect to the *Amiga* remotely via a secure shell (SSH) and any other WiFi-enabled peripherals.

Additionally, the Center for Agroecology has finger weeders, flexible fingers that uproot small weeds from plant base, and duck-foot cultivators, a hard metal spikes that stirs the top few inches of soil to remove thread-stage weeds ¹. These are vital towards maintaining and weeding unwanted plants at the UC Santa Cruz farm.

¹Very young newly germinated weed. Usually less than 1 inch long.

2.1 Kinematics

The 4x4 skid-steer drive train can be effectively modeled using a differential drive model, as shown in Figure 2.4.

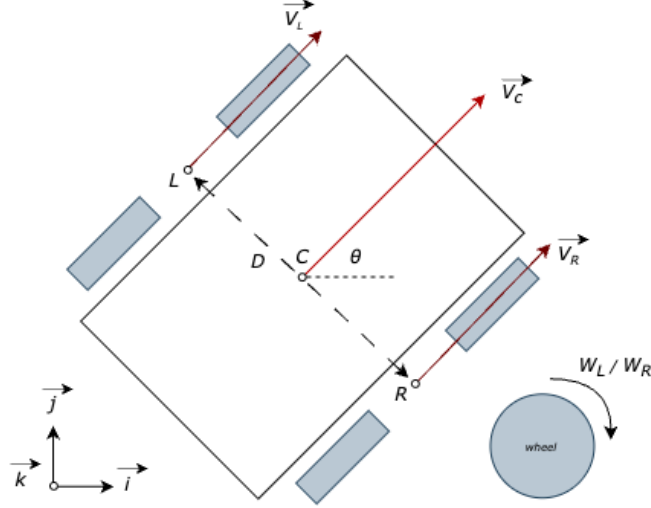


Figure 2.4: 4x4 skid-steer *Amiga* kinematic schematic: $\vec{i}, \vec{j}, \vec{k}$ denotes the global inertial frame unit vectors; L, R are the combined left and right wheel points; $\vec{V}_L, \vec{V}_C, \vec{V}_R$ are the velocity vectors at the left wheel, vehicles center C , and right wheel respectively; W_L/W_R indicate the wheel angular speeds; the dashed line from L to R represents the distance D ; heading angle is represented as θ ; and $\omega = \dot{\theta}$ as the angular velocity of the robot about the \hat{k} axis (out of the plane).

To simplify the model above, the left and right motor pairs can act as a single motor, enabling a differential drive model of the *Amiga* kinematics. Each wheel is assumed to have a radius r and angular velocities of ω_L and ω_R respectively. Concretely,

the speed of each wheel is defined as (2.1).

$$v_L = \omega_L r, \quad v_R = \omega_R r \quad (2.1)$$

Furthermore, the maximum velocity v and turning rate of the *Amiga* is defined as Equation 2.2.

$$v, \omega \in [-1, 1] \quad (2.2)$$

The velocity of each wheel, relative to the center of *Amiga*, depends on its linear velocity \vec{v}_C , and the angular velocity ω , through the relation:

$$\vec{v}_R = \vec{v}_C + \omega \hat{k} \times \vec{r}_{R/C} \quad (2.3)$$

$$\vec{v}_L = \vec{v}_C - \omega \hat{k} \times \vec{r}_{L/C} \quad (2.4)$$

where $\vec{r}_{R/C}$ and $\vec{r}_{L/C}$ are the position vectors from the center point C to the right and left wheels respectively. These vectors are orthogonal to the heading direction and have magnitude $\frac{D}{2}$:

$$\vec{r}_{R/C} = -\frac{D}{2} \vec{j}_R, \quad \vec{r}_{L/C} = \frac{D}{2} \vec{j}_R \quad (2.5)$$

Note that these two equations are equal in magnitude and opposite in direction.

$$\vec{r}_{R/C} = -\vec{r}_{L/C} \quad (2.6)$$

To isolate the linear velocity v_C , summing the equations (2.3) and (2.4) cancels out the angular terms resulting in:

$$\vec{v}_R + \vec{v}_L = 2\vec{v}_C \quad (2.7)$$

Now, to isolate its angular velocity, subtracting the equations (2.3) and (2.4) eliminates the linear velocity terms:

$$\vec{v}_R - \vec{v}_L = 2\omega\hat{k} \times \vec{r}_{R/C} \quad (2.8)$$

Now, substituting (2.5) into (2.9) and taking the cross product gives:

$$v_R - v_L = \omega D \quad (2.9)$$

Solving for v_C and ω in terms of the velocities of the left and right wheels:

$$v_C = \frac{v_R + v_L}{2} \quad (2.10)$$

$$\omega = \frac{v_R - v_L}{D} = \dot{\theta} \quad (2.11)$$

These set of equations model the movement of the *Amiga* tractor. With the introduction of a virtual joystick, they become vital to map the joystick's position to the tractors turning rate and velocity.

2.2 Software

The Python scripted Amiga Software Development Kit (SDK) by Farm.ING, provides developers access to each of the tractors services. With their own line of communication, each service controls a subsystem of the tractor. The main service, known as the *The Amiga Service*, acts as the central hub to: *OAK* cameras, *GPS*, and *Recorder*. Other available services include *CAN bus*, for motor connectivity, *Filter*, for state estimation, and *Track Follower* for point-to-point navigation. For the purposes of mobile interaction, the *The Amiga Service* and *CAN bus* services were primarily used.

The *OAK* sub-service provides a real-time camera stream from the on-board *OAK* cameras connected to the PoE switch. To receive video feed requires subscribing to a specific camera stream endpoint (URI), such as “rgb”, “left”, or “disparity,” at a specific polling rate. *TurboJPEG* then decodes the byte payloads into image frames. Eventually these frames are used in conjunction with navigation and communication protocols within the mobile application. Algorithm 1, depicts the provided Farm_NG software to correctly subscribe, gather, and extract frame messages from the OAK cameras [2]. Implemented on the *Amigas* main software, this functionality is within a function named *stream_camera()*.

Algorithm 1 Service subscription pseudocode

```

async for event, payload in canbus_client.subscribe(
    SubscribeRequest(uri=Uri(path="/state"), every_n=rate),
    decode=False)

```

The *CAN bus* service allows control over the robot’s motors, allowing developers to transmit motor states and send specific commands. Through a Twist2D² message format, the developer is able to command the linear and turning rate velocity of the tractor at 50 Hz. Within the *Amigas* main software, this functionality is implemented within a function named *pose_generator()*. In order to run services concurrently on the *Amiga*, two steps are required. First, each service client must be initialized with meta-

²A common pose command format to actuate a differential drive robot.

data specifics, such as its name, URIs, and polling rates within the *service_config.json* file. Once defined, the main Python application creates tasks using the *EventClient* class provided by the SDK, driven by Python's library *asyncio*. For instance, functions *stream_camera()* and *pose_generator()* allow for polling and service routines at a set rate. This framework also allows developers to include their own services and asynchronous functions. For the purposes of this thesis, additional functionality was integrated for a concurrent client-server architecture, as outlined within in Chapter 3. The creation and initialization of tasks are within a function named *app_func()*.

Chapter 3

Communication

To ensure that features on the mobile application can be utilized with the *Amiga*, a fast and reliable communication link is required. Several communication methods were considered:

- **Wired connection:** via USB or the on-board PoE was ruled out, as the core goal of this thesis was to enable remote operation between the user and the tractor.
- **Bluetooth:** while convenient for short-range communication, was not a viable option. The *Amiga* does not support Bluetooth without an external peripheral. Additionally, Bluetooth has limited range.
- **Internet based communication:** is able to leverage commonly used protocols over a defined WiFi connection. Connecting the *Amiga* to a hot-spot generated by the mobile device enables the user to create a direct connection without additional hardware. This method was utilized in communication efforts between the *Amiga*

and mobile devices.

Having determined a focus towards internet traffic as the line of communication, a protocol had to be adapted. Two widely used protocols were considered:

- TCP (transmission control protocol): The TCP is a commonly used connection-based protocol suited for high reliability. It ensures that data packets arrive intact and in the correct order through error checking and recovery operations. In a TCP connection, the server and client first establish connection through a handshake, where they are then able to send and receive data with confirmation receipts (ACKs). One drawback to this method is its speed of transaction between server and client because of its extra processing requirements [9].
- UDP (user datagram protocol) : The UDP is a connection-less based protocol. In opposition to TCP, the server and client do not establish a secure connection. Instead, the server acts as a hub where clients send datagrams to. These datagrams contain the address of the sender, i.e. client, which the server then uses to send data back, if required. Typically used in video streaming, this method enables high-speed transfer of large amounts of data. Additionally, it allows multiple clients to connect to the same server without much latency. The main drawback is its lack of security in messages. [9]

To facilitate the exchange of data, a server-client architecture was implemented as depicted in Figure 3.1. The *Amiga* stood as the server, while mobile devices acted as the clients. This structure allows the server to broadcast any messages, status updates,

and video feed to all connected clients. In turn, clients can connect to the server and send direct commands to the *Amiga*. To ensure functionality, the server is responsible for handling client logic, such as timeouts, and which mobile device is in control of the *Amiga* at that moment in time.

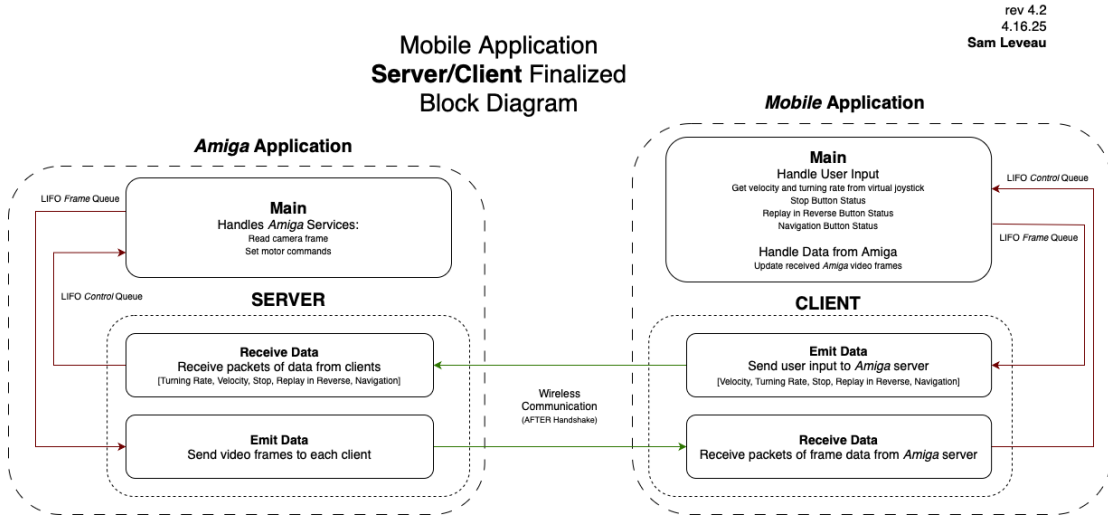


Figure 3.1: Server/Client Communication Block Diagram Structure

Communication between server and client is established through the use of sockets¹, binding the device's IP addresses to its port. Once the mobile device and *Amiga* are connected to the same local hot-spot, the server (acting as a central hub) creates a socket bound to its local IP address and specified port. In contrast, the client binds its local IP address to a random available port. The client then uses the known IP and port address of the server to send datagram packets. Both the server and client require initialization of the correct communication model. In Python, this is done with the socket library and Algorithm 2. *AF_INET* specifies the IPv4 address family and

¹The doorway for processes to send and receive message through [9].

SOCK_DGRAM initializes the use of the datagram protocol. In addition to creating and binding to an IP address, each socket sets a timeout - ensuring that the server or client does not block other processes while waiting for a message, and buffer size - specifying the amount of data that it is able to receive.

Algorithm 2 Initialization of a *socket* in Python

```
socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

3.1 Server

In order to host a server on the *Amiga*, two asynchronous tasks were implemented through the provided *Amigas* SDK. More specifically, using Python's *asyncio* library, defined functions *recv()* and *send()* - encapsulated within a newly made Python class - were responsible for dealing with new client connections, incoming data, and sending datagram packets.

To ensure data transfer between the communication tasks and the *Amigas'* main processes, Python's *Queue* library was utilized. Two last-in-first-out (LIFO) buffers were created for frame data (*frame*) and mobile application commands (*command*). Once a new frame was available in the *Amigas'* main process, a copy of it was placed into the *frame* queue; when *send()* next ran, it dequeued that frame and transmitted it as a datagram to all connected clients. Likewise, when incoming mobile application commands were received, it was queued into the *command* queue, allowing

the *Amigas* main process to pull the latest command from the joystick and navigate accordingly. The following Algorithms outlines the basic structure of each function.

Algorithm 3 Server *recv()* pseudocode

Remove inactive clients

for all client in connected clients **do**

 Remove client that hasn't sent a message a in over a second

end for

Check for new messages

Read the UDP socket for data without pausing other tasks

if client is new **then**

 Add client to the connected clients list and send a connection confirmation

end if

if cleint is already connected **then**

 Decide if this client becomes the *main* controller

 Update the client's timestamp and *main* status

 Place decoded commands into the LIFO queue for later use

end if

Algorithm 4 Server *send()* pseudocode

```

if no clients are connected then

    Break

end if

for all client in connected clients do

    Dequeue latest frame from frame LIFO, resize, compress it with TurboJPEG

    if client is main then

        Add an additional bit to relay main client in control

    end if

    Transmit the datagram packet to client

end for

```

3.2 Client

Similar to the server's structure, two separate functions, *recv()* and *send()* were created to achieve concurrent programming. However, instead of utilizing *asyncio*, the *threading* library was chosen because mobile operating systems are better suited to manage multiple threads efficiently. When the user attempts to connect to the *Amiga*, a connection request is attempted; once confirmed, the threads are initialized and started, as described in Algorithm 5. Likewise, once the user disconnected, they are stopped and removed. Additionally, two LIFO queues, *frame* and *command* were used to transfer

data between the applications main process and the threads.

Similar to the server, the following structure was utilized. Note: Algorithm 5 is an non-concurrent function.

Algorithm 5 Request Connection pseudocode

Create a UDP socket to any free local port, set buffer and timeout

Send handshake message to server

try:

Wait for new connection confirmation

Initialize and start threads

except socket timeout :

Break

Algorithm 6 Client *recv()* pseudocode

if cleint is connected to server **then**

Wait for any incoming packets of data

Parse the datagram packet

Queue the frame data to *frame* queue and set client to *main* if applicable

end if

Algorithm 7 Server *send()* pseudocode

```
if client is connected to server then  
    if command queue is not empty then  
        Dequeue command, encode and transmit it to the server  
    end if  
end if
```

3.3 Message Structure

The server transmits a combination of the *TURBO-JPEG* encoded image frames and a single bit indicating whether the receiving client is currently the *main* controller. On the other hand, the client sends control commands encoded as a string representing a list of five values:

- Turning rate (float)
- Velocity (float)
- Autonomous navigation (boolean)
- Emergency stop (boolean)
- Replay in reverse (boolean)

Chapter 4

Kivy

Kivy is an open-source framework designed for developing cross-platform graphical user interfaces (GUI) applications in Python. This includes deployment on Windows, macOS, Linux, iOS, and Android. Its structure is designed on modular building blocks, allowing developers to choose specific functionalities for different platforms. As seen in the Kivy Architecture, (Figure 4.1) developers are able to access low level functions and abstract them to high level software development.

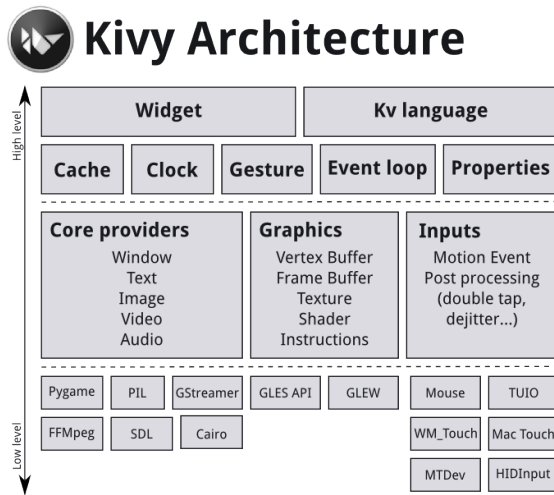


Figure 4.1: Kivy Graphical Architecture [7]

At its core, starting a Kivy application requires three main components: an instance of a Kivy defined App class, implementing the Apps build method, and running its internal looping. Built upon this, a custom Kivy application consists of Python logic that defines behavior and handles user input logic, and a Kivy language file (extension *.kv*) that outlines the user interface. Behind the scenes, the Kivy framework manages the event looping and user input.

To illustrate the simplicity behind running an application, Algorithm 8 outlines the pseudocode to run a Kivy application and display text using the label widget. It demonstrates the minimal setup required before adding helpful widgets and logic.

Algorithm 8 Basic Kivy Python Structure Pseudocode

Kivy Imports

App Class

def Build

return “Hello World” Label

Run App Class

Figure 4.2 depicts the resulting structure. It’s important to note that the Kivy language file isn’t used in this basic example. The main application can directly link to Kivy’s widget graphics.

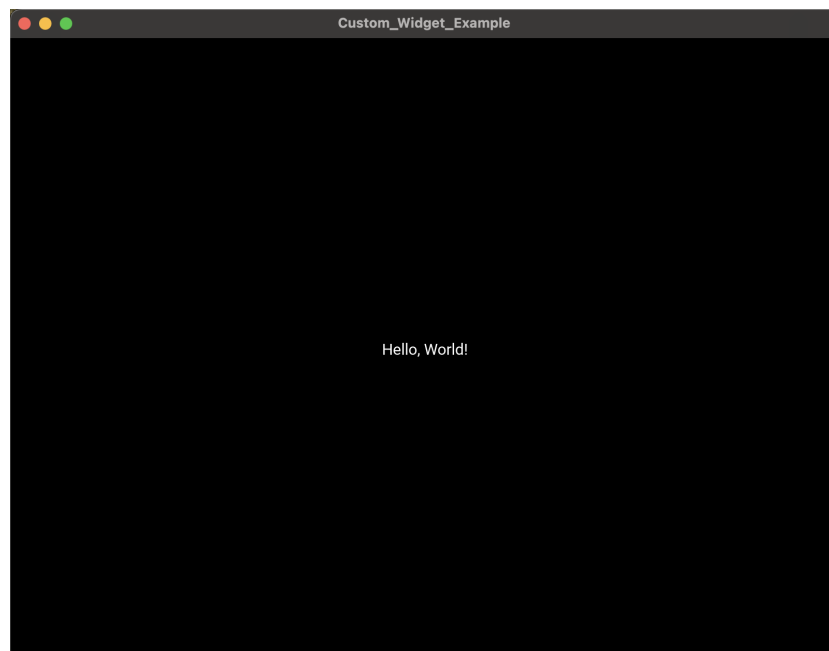


Figure 4.2: Hello World Example

Built into a hierarchical tree structure,¹ Kivys' widgets are the fundamental building blocks of the GUI interface. Each provide a Canvas that can be used to draw on screen and an event driven framework for any user input. While Kivy provides many built-in widgets, like buttons, sliders, and toggles, some do not meet the standard for user-friendliness and modularity for the requirements of this project. To achieve full control over the appearance and functionality of each user interaction, custom widgets were developed. This enabled the creation of a reusable toggle button, that can be instanced throughout the application, and a virtual joystick. In Python, the basic building block for the custom toggle widget is defined in Algorithm 9.

¹Every Kivy application contains a single root widget, which can contain child widget. Each of these children widgets can have their own children, forming a recursive widget tree that outlines the UI.

4.1 Custom Widgets

Algorithm 9 Toggle button class pseudocode

```

if User touch widget then

    Toggle current state

    Update button visual UPDATE_BUTTON_STATUS()

end if

function UPDATE_BUTTON_STATUS(status)

    if status is True then

        Update toggle button color to Green

    else

        Update toggle button color Red

    end if

end function

```

The accompanying Kivy language file handles the user-friendly appearance by drawing an ellipse and border whose fill color responds to its toggled state, on or off (Algorithm 10).

Algorithm 10 Toggle button class Kivy Language file pseudocode

Define an ID for the widget

Define global color-fill variables

Define the widgets size and position

Canvas Drawing:

Draw an ellipse with a border

Now, each time the user presses the button, the toggled property flips, updating the color (Figure 4.3) and triggering any other actions tied to that state.

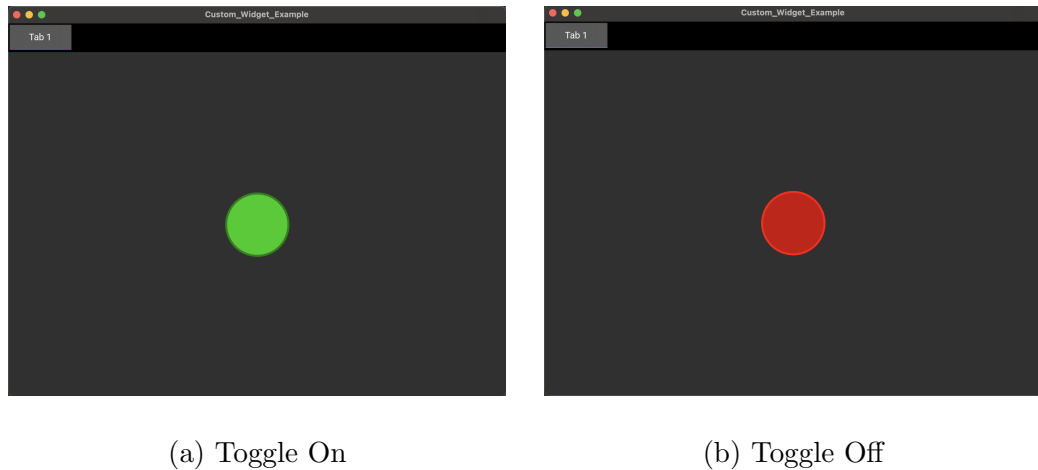


Figure 4.3: Custom Toggle Button in its two States

The limitation with the presented approach is that the widgets `on_touch_down` handler relies on the full square canvas rather than the depicted ellipse on screen. Thus, any tap within that defined canvas square, will trigger the event. To restrict this event

call, the standard cartesian equation of a circle was utilized.

4.2 Virtual Joystick

Similar to the toggle button, the virtual joystick is an instance of the custom widget class that captures user drag events and translates them into pose commands for the *Amiga*. Figure 4.4 and accompanying mathematical formulation describe how the joystick's movement is mapped to the velocity and turning rate commands. Recall the *Amiga*'s speed is limited to bounds, as outlined in Equation 2.2.

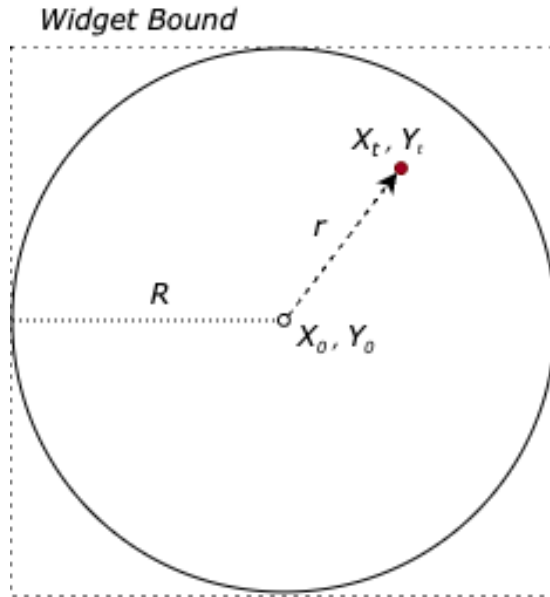


Figure 4.4: Joystick Widget Model

Let (x_t, y_t) be the touch coordinates and (x_0, y_0) be the joystick's center. The displacement vector components are as follows:

$$\Delta x = x_t - x_0, \quad \Delta y = y_t - y_0, \quad (4.1)$$

and its magnitude,

$$r = \sqrt{\Delta x^2 + \Delta y^2}. \quad (4.2)$$

If $r > R$, where R is the joystick's maximum radius, clamp the touch position back onto the circle:

$$(\Delta x', \Delta y') = \begin{cases} (\Delta x, \Delta y), & r \leq R, \\ (norm(r) * R), & r > R. \end{cases} \quad (4.3)$$

Then update the dynamic knob position to the new calculated position at Equation 4.4.

$$(x_0 + \Delta x', y_0 + \Delta y') \quad (4.4)$$

In order to convert the user's touch position into movement actions within the $[-1, 1]$ range, Equation 4.5 scales the input to the maximum velocity and turning rate and the joystick radius R . This allows for any re-sizing of the joystick without error. Figure 4.5 depicts the custom joystick widget centered and active.

$$v = \frac{\Delta x' * range(0, 1)}{R}, \quad \omega = \frac{\Delta y' * range(0, 1)}{R}. \quad (4.5)$$

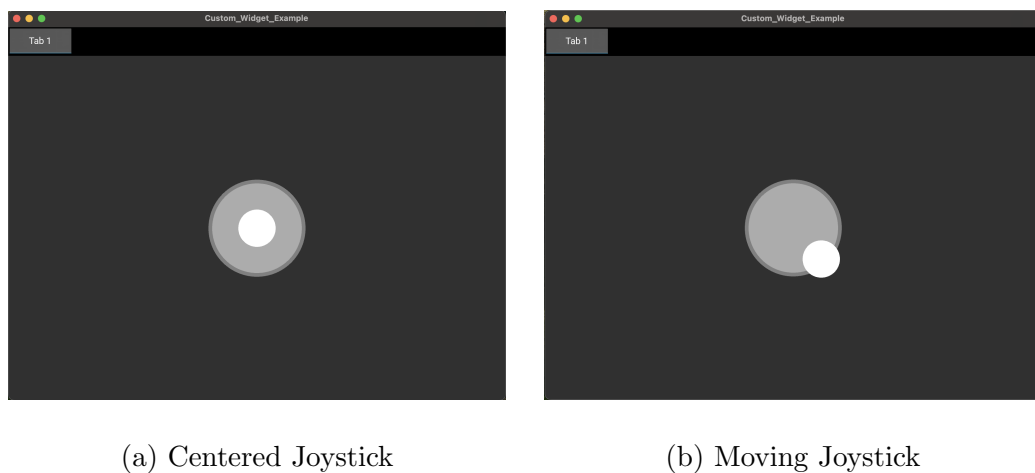


Figure 4.5: Virtual Joystick

It is important to note that according to Figure 4.4, inherent to the circulars circumference, the maximum velocity and maximum achieved at the same time is limited to values $[-0.7, 0.7]$.

Chapter 5

Mobile Application

The full development of the mobile application system and the accompanying *Amiga* tractor comprises two primary layers:

- **Mobile Device:** packaged for Android and iOS using Buildozer [8] — which includes a Kivy-based GUI and its concurrent client.
- **The *Amiga*** — its main process, combining the existing plant-line following autonomous navigation with the newly developed server.

5.1 The *Amigas* Main Process

To easily implement the mobile application commands to the *Amigas* main process, a state machine structure was added. This organized the velocity and turning rate commands sent to its CAN bus network. The following states were implemented.

- **IDLE:** No client connected

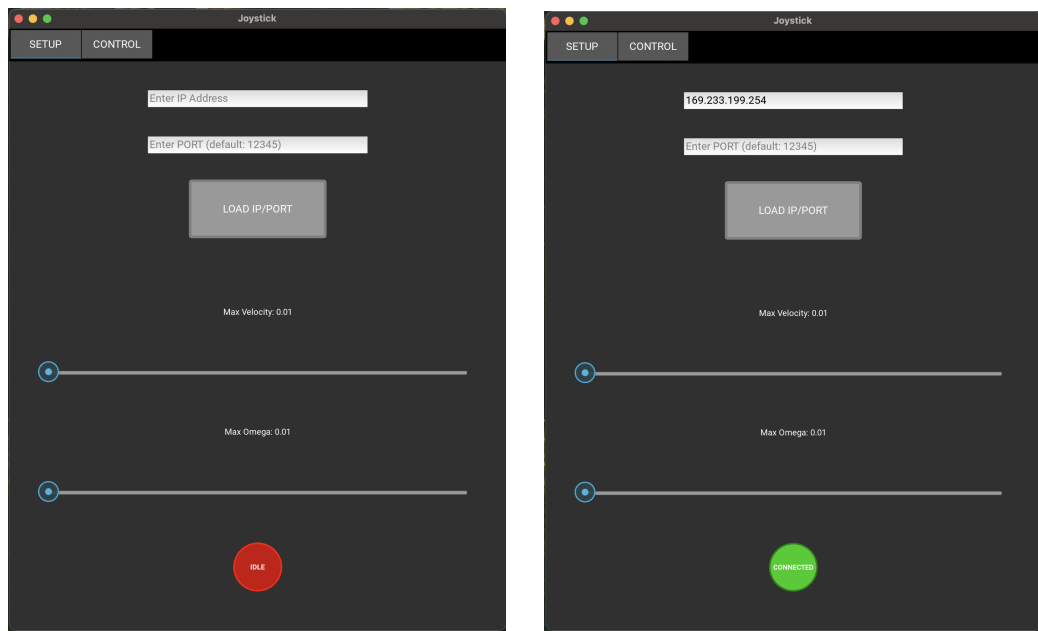
- JOYSTICK: The mobile app is connected; sets the incoming joystick velocity and turning rate commands
- NAVIGATION: Set calculated autonomous navigation controller velocity and turning rate. If no mobile device is connected, this can be toggled from the *Amigas* main application; if a mobile device is connected, it can only be toggled via the application.
- STOP: Emergency stop
- REVERSE: Replay-in-reverse

5.2 Graphical User Interface

The mobile application's graphical user interface is organized into two tabs: SETUP and CONTROL. The SETUP tab enables the user to establish a server connection and configure control parameters, while the CONTROL tab provides operation tools, such as a virtual joystick, emergency stop, autonomous control toggles, and reverse-in-replay. The design and functionality of each tab are described in detail below.

5.2.1 SETUP Tab

As depicted in Figure 5.1, the first tab under the application allows the user to connect to the established server created on the *Amiga* and set the maximum velocity and turning rate (both normalized to $[0,1]$ range, see Equation 2.2).



(a) SETUP Tab Not Connected

(b) SETUP Tab Connected

Figure 5.1: SETUP Tabs

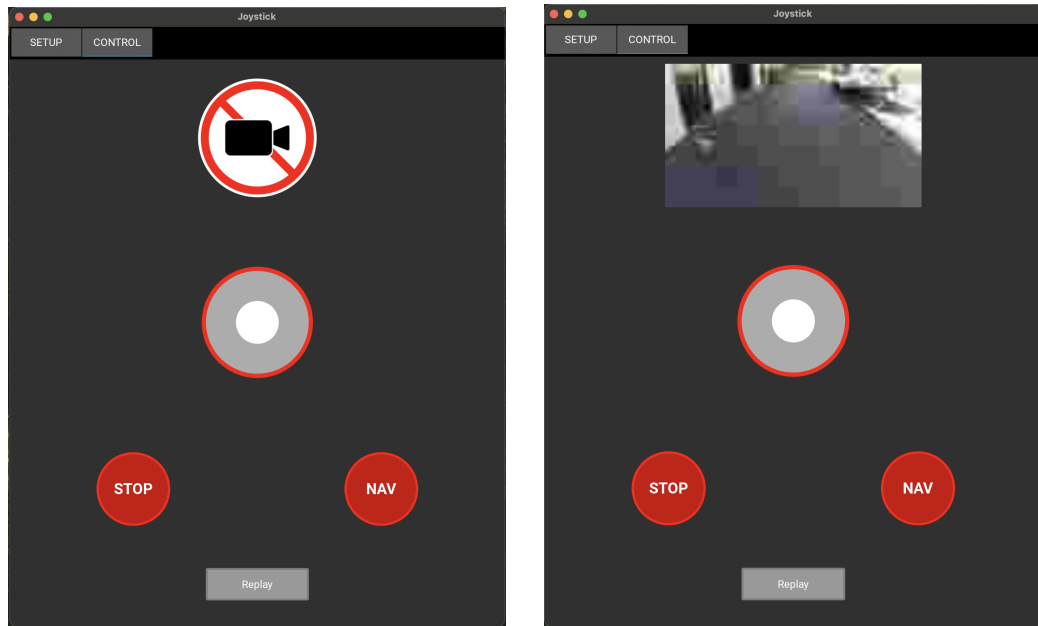
The SETUP tab components include:

- Kivy TextInput widgets were utilized for inputting the server IP and port information
- Kivy Slider widgets were used to set the range of speeds
- Kivy Button widget used to load IP and PORT values
- Connect Button: An instance of the custom widget developed from the Kivy Chapter (Figure 4.3) was created to serve as the connect toggle button with an additional status label. When pressed, the non-asynchronous function outlined in Algorithm 5, is executed to establish the UDP socket connection to the *Amiga*

server. If the connection succeeds, the button is toggled green. Once toggled again, the client disconnects and terminates the *recv()* and *send()* threads. If at any point the socket timeouts, the mobile application assumes the server has unexpectedly gone offline and toggles the button back to the disconnected state.

5.2.2 CONTROL Tab

The second tab, CONTROL (Figure 5.2), allows for operator control of the *Amiga*. The screen depicts the *Amigas* live video stream, virtual joystick, emergency stop button, autonomous navigation toggle button, and a button for replay-in-reverse.



(a) CONTROL Tab Not Connected

(b) CONTROL Tab Connected

Figure 5.2: CONTROL Tabs

More specifically, components include:

- Kivy Texture was utilized for setting a new *Amiga* video frame. If no client is connected, the texture is set to a default image as seen in Figure 5.2a.
- Virtual Joystick: Developed as a custom widget in the Kivy Chapter, this Joystick allows for velocity and turning rate commands of the *Amiga*. Note that the JOYSTICK is the *Amigas'* default state if a mobile application is connected.
- Emergency Stop: Similar to the Connect button in the SETUP tab, an instance of the custom toggle widget was created with an additional label status. When toggled on, the mobile application blocks all other user input and immediately transmits the velocity and turning rate to zero. Compared to the physical emergency e-stop as depicted in Figure 2.3, this method of braking prevents any roll of the *Amiga*. Only after disabling the button will the user gain functionality of the *Amiga* again.
- Autonomous Navigation: With another instance of the custom toggle created, this button controls the navigation status of the *Amiga*. Once activated, the *Amiga* transitions to the NAVIGATION state and utilizes the velocity and turning rate commands generated by its on-board controller. Additionally, all commands are recorded into a set buffer, later used for the replay-in-reverse functionality.
- Replay-In-Reverse: For autonomous debugging purposes and operator correction, this feature allows the user to reverse the most recent autonomous navigation movement by a set buffer size (as discussed in the autonomous navigation point above). Unlike the toggle feature, the user must press and hold the button for

timer of their choosing. If activated the *Amiga* transitions to the REVERSE state and begins dequeuing from the buffer, negates these control actions, and sends these commands to the CAN bus network. Once the buffer has been emptied the next command is waited upon.

5.3 Typical Workflow

In order to activate and use the fully developed mobile application and the accompanying *Amiga* system, the operator must follow three main steps: (1) Establish Connection & Initialization; (2) CONTROL Tab Operation; (3) Disconnect & Shut-down. Each of these has more incremental steps as described below.

Step 1: Establish Connection & Initialization

- 1.1: Begin local hotspot on mobile device
- 1.2: Power on the *Amiga*, connect to the hotspot, launch its main application and set it to AUTO CONTROL
- 1.3: Input and LOAD the *Amigas* IP address into the SETUP tab (server IP is displayed on the *Amigas* Dashboard)
- 1.4: Adjust the velocity and turning-rate sliders to define maximum Joystick allowable commands

Step 2: CONTROL Tab Operation

- 2.1: Switch to the CONTROL Tab to access the live video feed and control widgets.
- 2.2: Drag the virtual joystick to send movement commands
- 2.3: Toggle the emergency stop button to halt all action and lock other inputs
- 2.4: Toggle the NAV button to use plant-line autonomous navigation
- 2.5: Press and hold the Replay button to play back and invert the autonomous navigation
- 2.6: If desired, the operator is able to navigate back towards the SETUP tab and alter the slider parameters

Step 3: **Disconnect & Shutdown**

- 3.1: When finished, toggle the Connect button to disconnect from the server

5.4 Results

To evaluate the performance of the mobile application, especially at its most commonly used function, the Joystick, its responsiveness (latency) and functionality (operator effectiveness) were evaluated.

5.4.1 Responsiveness

To measure the system's responsiveness, the time delta between sending a movement command on the mobile application, and observing the *Amigas'* movement

was measured. Each trial consisted of accelerating the *Amiga* from rest (0 m/s) to 0.5 m/s and then decelerating back to rest. A video was recorded of both the mobile application and the *Amigas* motion, allowing precise determination of latency. Summarized in Table 5.1, the average round-trip latency across the ten trials was 0.46 seconds.

Trial	Delta Time (seconds)
1	0.4
2	0.4
3	0.4
4	0.5
5	0.4
6	0.6
7	0.5
8	0.4
9	0.4
10	0.6

Table 5.1: Measured round-trip latencies for ten trials.

5.4.2 Functionality

Operator control effectiveness was quantified by measuring the time it took to run through an obstacle-course consisting of a 2 meter straight run, U-turn around a cone and 2 meter return to the start line as depicted in Figure 5.3.

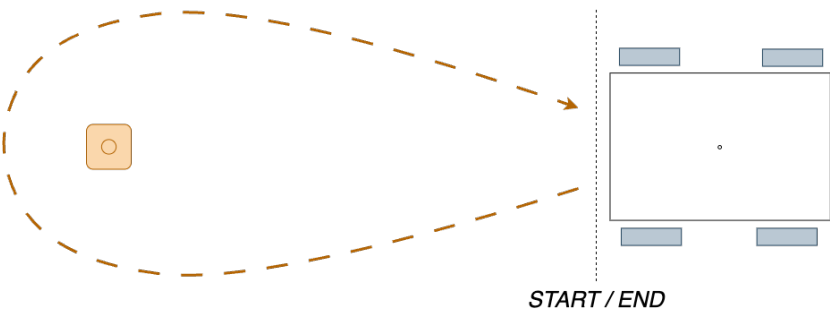


Figure 5.3: Functionality Trial Run Configuration

Three control modes were tested; the pendant (as a baseline), remote joystick with the *Amiga* in line of sight, and remote joystick control solely relying on the live video feed. Table 5.2 depicts the results of these trials.

Tethered joystick	Remote joystick with sight	Remote joystick with video
20 seconds	30 seconds	40 seconds

Table 5.2: Obstacle course completion times across control modes.

The pendant showed the fastest completion time (as expected) while remote operation via video stream was the slowest. Despite the increased time, all runs were completed successfully without any collision.

Chapter 6

Conclusion

Over the past few decades, the California agricultural industry has been grappling with labor shortages, safety hazards, and long-term degradation of soil health. To address these challenges, this senior thesis project aimed to develop a mobile application, increasing the capabilities of the organic driven electric tractor: the *Amiga*. Structured with a Kivy framework and integration with a UDP communication protocol, the application provides remote control functionality through a virtual joystick, live video feed, and command toggles. These elements take a step towards a more accessible, sustainable, and safer agricultural practice, empowering organic farmers with a tool that reduces exposure to physically demanding tasks and dangerous conditions. Additionally, the mobile app bridges the vital human-machine interaction gap by enabling intuitive control of the *Amiga*.

6.1 Future Improvement

As described in Chapter 5, the current implementation of the autonomous navigation requires initialization on the *Amigas* Dashboard. A future iteration of the mobile application could include a tab for configuring these parameters directly from the user's device, completing the transition to full remote operation.

Another manual process that is vital to the research being completed in the Robotics and Control lab is data collection. Currently, a researcher is required to find the collected data file within the *Amigas* storage and copy it locally. To avoid this hassle, a new mobile application tab can be created with choices of data collection. That is, once the operator is ready, they would be able to stop the *Amiga* and transfer desired data straight to their mobile device.

After discussion with the manager of the Center for Agroecology, there was interest in developing a remote physical joystick for controlling the *Amiga*. For example, a PlayStation controller paired with a mobile phone, could be used to operate the tractor from a distance. The modular server-client architecture would enable this integration.

6.2 Personal Reflection

This senior thesis project has significantly advanced my understanding of embedded systems, communication protocols, and human-robot interaction. More importantly, it has deepened my awareness on the implications of implementing technology within interdisciplinary contexts. Throughout the work, I honed in on my technical

skills while continuously engaging with peers and faculty to ensure that each feature addressed a genuine operator need. This experience has not only fostered lasting professional relationships, but also instilled a commitment to developing technical solutions that are both effective and ethically responsible.

Bibliography

- [1] California Department of Food and Agriculture. California agricultural production statistics, 2024. Accessed: April 29, 2025.
- [2] Farm_NG. Farm_ng official website. Accessed: April 29, 2025.
- [3] Francisco Soto Mas, Alexis J. Handal, Rose E. Rohrer, and Eric Tomalá Viteri. Health and safety in organic farming: A qualitative study. *Journal of Agromedicine*, 23(1):92–104, 2018. Accessed: April 29, 2025.
- [4] Linda Geist. Farming: The most dangerous job in the u.s., 2022. Accessed: April 29, 2025.
- [5] Victoria Graham. Ftc, states sue deere and company to protect farmers from unfair corporate tactics and high repair costs, 2025. Accessed: April 29, 2025.
- [6] Matt Hopkins. 10 new mobile apps for precision agriculture, 2017. Accessed: April 29, 2025.
- [7] Kivy Developers. Kivy, 2025. Accessed: April 29, 2025.
- [8] Kivy Organization. *Buildozer Documentation*. Read the Docs, 2025.

- [9] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 7 edition, 2016.
- [10] Mohd Ghazi R, Nik Yusoff NR, Abdul Halim NS, Wahab IRA, Ab Latif N, Hasmoni SH, Ahmad Zaini MA, Zakaria ZA. Health effects of herbicides and its current removal strategies. *National Library of Medicine*, 14(1), 2023. Accessed: April 29, 2025.
- [11] Karen Nikos-Rose. California farmers have raised wages, but still can't find enough workers, 2019. Accessed: April 29, 2025.
- [12] Iqbal Pittalwala. Southern california's farmworkers lack health resources, study says, 2023. Accessed: April 29, 2025.
- [13] Zachariah Rutledge and J. Edward Taylor. California farmers change production as the farm labor supply declines, 2019. Accessed: April 29, 2025.
- [14] Wikipedia Contributors. Driverless tractors, 2024. Accessed: April 29, 2025.