User Manual
Pseudo GPS for Romi System

Team Members:
Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott

ME Department, California Polytechnic State University San Luis Obispo

December 8, 2024

Abstract

This user manual offers comprehensive guidance for Cal Poly Mechatronic students on setting up, operating, and maintaining the Pseudo GPS for the Romi system. Utilizing a ceiling-mounted camera, the system tracks and monitors the positions and orientations of Romi robots within the ME mechatronics lab environment.

1 Bill of Materials

Romi Budget Plan	
Part	Cost $
Micro HDMI to HDMI cable	8.95
Raspberry Pi Camera Module 3 Standard - 12MP Autofocus	25
Raspberry Pi 4 model B-8GB RAM x2	150
Official Raspberry Pi Power Supply 5.1V 3A with USB C	7.95
SD/MicroSD Memory Card - 16GB Class 10 - Adapter Included	19.95
Official Raspberry Pi 4 Case Fan and Heatsink	5
Twozoh 4K micro HDMI to HDMI 60Hz 1080p	13.99
USB CHarger Block OKRAY 2-pack 18W	8.99
Baiwwa USB C Cable, USB A to type C Cable Charger	8.98
Parts Total	248.81
Initial Budget	500
Current Budget	251.19

Recommended Parts	
Part	Cost $
Raspberry Pi Screen, ROAMDOM 10.1" Touchscreen Monitor, 1024 x 600	99.99
Logitech MK270 Wireless Keyboard and Mouse COmbo	27.99
Beikell SD Card Reader, Dual Connector USB C Memory Card Reader Adapter	7.59
Parts Total	135.57
Current Budget	251.19
New Budget	115.62

Figure 1: Bill of Materials Image

2 Software Setup

This section explains how to set up the software environment on the Raspberry Pi to run the Pseudo GPS for Romi system. It covers how to turn on the Raspberry Pi, install required libraries like OpenCV, and run the provided Python code to interact with the camera and perform ArUco marker detection.

2.1 Required Python Imports

To run the system, you'll need to install and import several Python libraries. Below are the imports required for the Thonny Python environment and the code functionality.

2.1.1 Required Imports

Include these imports at the beginning of your Python script:

```
import cv2 # For image processing and ArUco marker detection
import numpy as np # For handling arrays and matrix operations
from picamera2 import Picamera2, MappedArray  # For accessing the
    Raspberry Pi camera
import time # For managing time-related operations
import math # For mathematical operations like trigonometry
import os # For handling system-related operations
import sys # For system-specific parameters
import threading # For managing concurrent operations
import argparse # For parsing command-line arguments
from cv2 import aruco # For working with ArUco marker detection
```

These imports are necessary for the core functionalities of the Pseudo GPS system:

- `cv2`: OpenCV, used for image processing and computer vision tasks like ArUco marker detection.
- `numpy`: For handling matrix and array-based computations.
- `picamera2`: To interface with the Raspberry Pi camera.
- `time`: For managing delays or intervals, like timing for marker detection.
- `math`: For handling mathematical operations such as angles or trigonometry.
- `os`, `sys`: For system operations, managing file paths, and handling the environment.
- `threading`: If you're working with concurrent tasks.
- `argparse`: For parsing command-line arguments if necessary.
- `aruco`: For specific functions related to ArUco markers.

3 Installing Dependencies

To run the provided Python code, you will need to install OpenCV, Picamera2, NumPy, and other libraries. Below are the installation steps.

3.1 Install OpenCV and Required Libraries

Use the terminal on your Raspberry Pi to install the necessary libraries.

3.1.1 Update the Raspberry Pi

To Update the Pi, you may need this info:

username for Pi: emanbz909
Password: BigRed405
this is for main Pi in black housing clone of pi is the white housing
username for clone Pi: emanbz909 (@CodyClone1)

Open a terminal on the Raspberry Pi and run the following command to update the package lists:

```
sudo apt-get update
sudo apt-get upgrade
```

3.1.2 Install Dependencies

Install the required dependencies for OpenCV, camera, and additional libraries:

```
sudo apt-get install python3-opencv
sudo apt-get install python3-picamera2
sudo apt-get install python3-numpy
sudo apt-get install python3-matplotlib
```

3.1.3 Install OpenCV via pip (if not already installed)

If OpenCV is not already installed via the default repositories, you can install it using pip. Note: This may take around one hour to install.

```
pip3 install opencv-python
pip3 install opencv-contrib-python
```

4 Hardware Setup

This section explains how to physically set up the hardware components of the Pseudo GPS for Romi system. This includes assembling the ceiling-mounted camera, installing the Raspberry Pi, and other components, and how the system works together to track the Romi robots.

4.1 Hardware Components

- **Raspberry Pi 4 Model B (1X)**: The main processing unit responsible for controlling the system and processing camera data.
- **Raspberry Pi Camera Module 3 (1X)**: A high-resolution camera mounted on the ceiling to capture the environment and track the Romi robots.
- **ESP32 Microcontroller (2X)**: Used for wireless communication between the Raspberry Pi and the Romi robots via WiFi.
- **Mounting Hardware**: Includes screws, threaded inserts, and toggle bolts for securely attaching the system to the ceiling.

4.2 Setting Up the Camera

4.2.1 Camera Installation

- Mount the Raspberry Pi Camera Module 3 securely to the ceiling using the camera housing.
- Ensure that the camera is properly aligned to provide an optimal field of view of the area where the Romi robots will be tracked.
- The camera is connected to the Raspberry Pi through the CSI (Camera Serial Interface) port.

4.2.2 Camera Calibration

The camera calibration process uses ArUco markers placed in the environment. The system estimates their pose (position and orientation) relative to the camera using the calibration matrix and distortion coefficients. You will need to adjust the **calibration factor** (located in our code) if the detected distances between markers are incorrect. This calibration must be done each time you change the camera's height distance from the arUco markers.

Resource:

- Camera Calibration Example: OpenCV Interactive Calibration

4.3 Hardware Setup Visual Images

Here are placeholder images to represent the hardware components. These images will be replaced with actual pictures once they are available.

4.3.1 Raspberry Pi 4 Model B with Mounting Hardware

Figure 2: Raspberry Pi 4 Model B in mounting hardware.

4.3.2 Raspberry Pi Camera Module 3

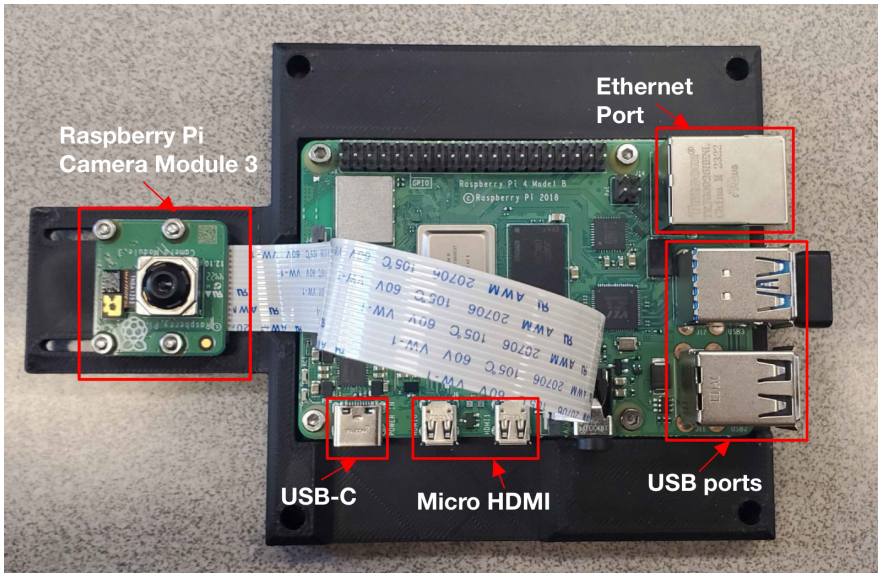

Figure 3: Labeled Raspberry Pi Camera Module 3 Top View

4.3.3 Raspberry Pi Camera Module 3

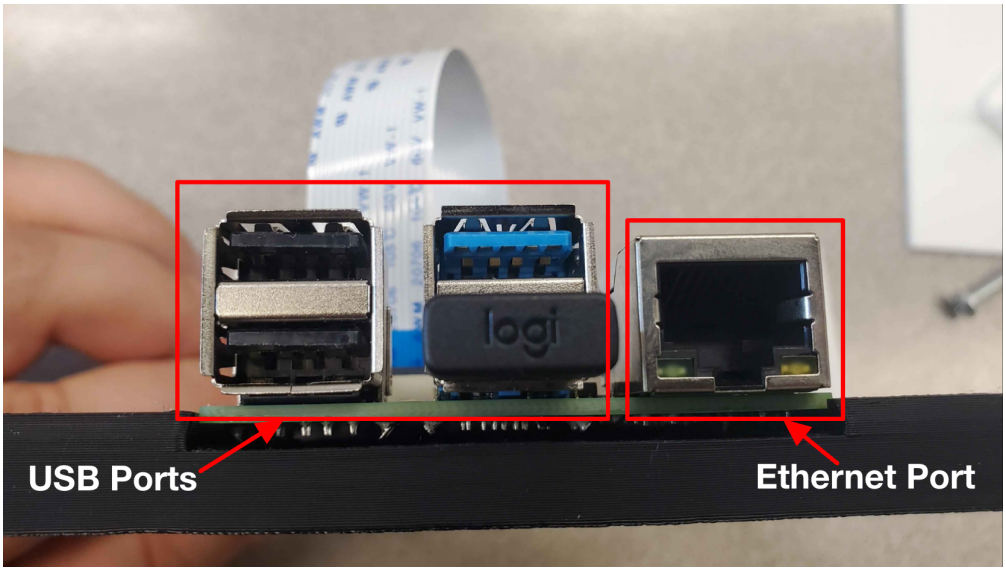

Figure 4: Labeled Raspberry Pi Camera Module 3 Back View

4.3.4 Raspberry Pi Camera Module 3

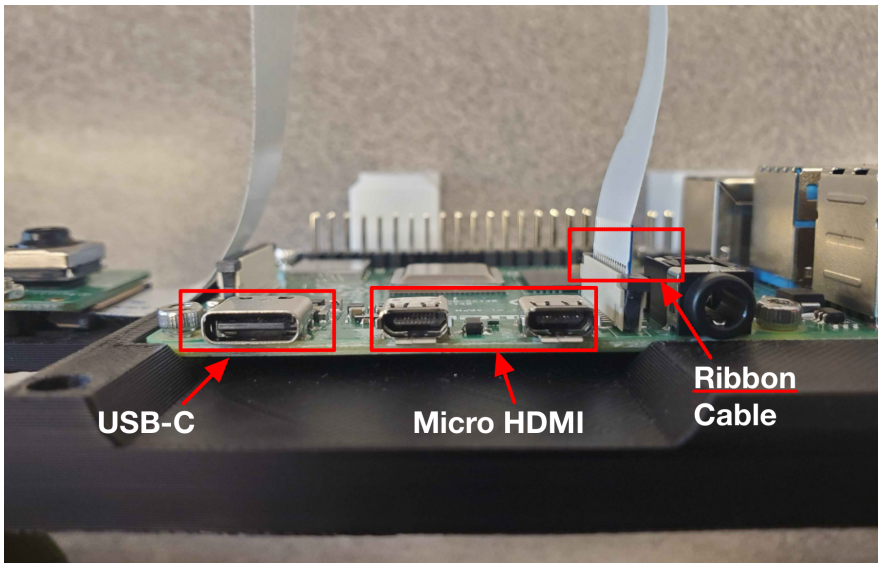

Figure 5: Labeled Raspberry Pi Camera Module 3 Side View

4.3.5 Raspberry Pi Camera Module 3

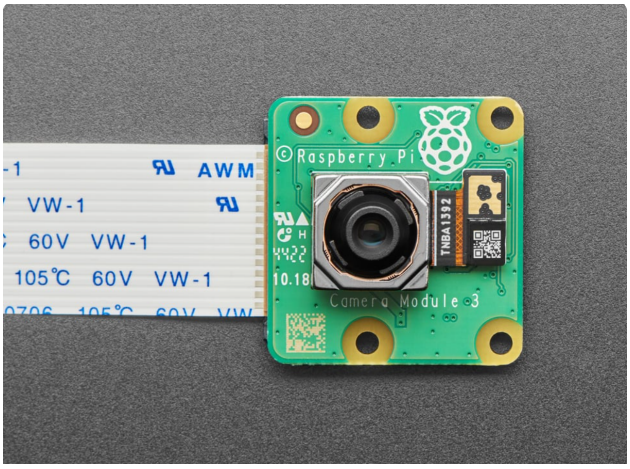

Figure 6: Raspberry Pi Camera Module 3

4.3.6 ESP32 Microcontroller

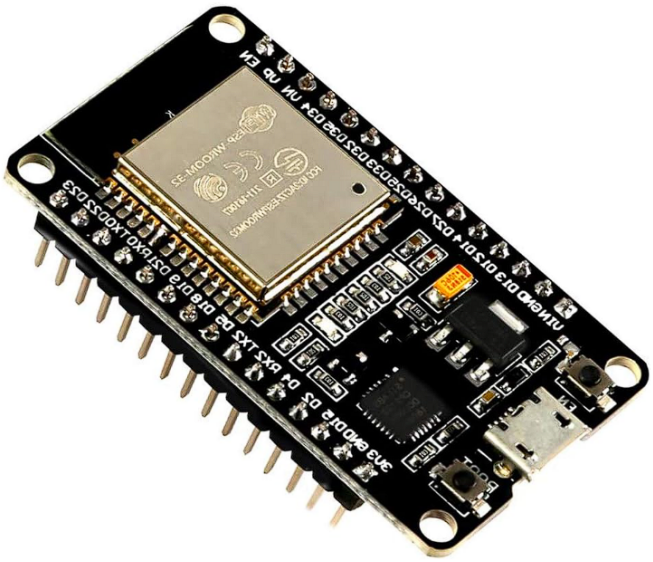

Figure 7: ESP32 Microcontroller

5 Code Walkthrough

Below is an explanation of the provided Python code, which runs on the Raspberry Pi to track and monitor the Romi robots using ArUco markers. The code performs the key functions of capturing images from the camera, detecting ArUco markers, and estimating their pose (position and orientation). Note: the orientation and position codes will not run until an ArUco marker is placed. The marker generator code only generates markers.

5.1 Initial Setup and Camera Class

```
import cv2
import numpy as np
from picamera2 import Picamera2, MappedArray
import time
import math
import os
import sys
import threading
import argparse
from cv2 import aruco

# Initialize the camera
class Camera:
    def __init__(self):
        self.picam2 = Picamera2()
        self.picam2.start()

    def capture_frame(self):
        frame = self.picam2.capture_array()
        return frame
```

5.1.1 Explanation:

1. **Imports**: The code begins by importing the necessary libraries. These include OpenCV (`cv2`), NumPy (`np`), Picamera2 for interfacing with the Raspberry Pi camera, and others for threading and argument parsing.

2. **Camera Class**: - A `Camera` class is defined that initializes the Pi camera (`Picamera2`) and starts capturing frames.

- The method `capture_frame()` captures a single frame from the Raspberry Pi camera and returns it as a NumPy array for further image processing.

5.2 Marker Detection and Pose Estimation

```
# Marker detection and pose estimation
def detect_markers(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    detector = aruco.ArucoDetector()
    markers, ids, _ = detector.detectMarkers(gray)
    return markers, ids
```

5.2.1 Explanation:

1. **detect_markers()**: - Converts the captured frame to grayscale using `cv2.cvtColor`. - Initializes an ArUco marker detector using `aruco.ArucoDetector()`. - Detects ArUco markers in the grayscale image using the `detectMarkers()` method, which returns the marker positions (`markers`) and their IDs (`ids`).

5.3 Main Program Loop

```
# Main program loop
if __name__ == "__main__":
    camera = Camera()
    while True:
        frame = camera.capture_frame()
        markers, ids = detect_markers(frame)

        if markers:
            for marker in markers:
                cv2.drawContours(frame, marker, -1, (0, 255, 0), 2)

        cv2.imshow("Camera Feed", frame)
        if cv2.waitKey(1) == ord('q'):
            break

    cv2.destroyAllWindows()
```

5.3.1 Explanation:

1. **Main Loop**:
   - The camera captures a frame in an infinite loop (`while True:`).
   - For each frame, the code checks for ArUco markers and draws contours around the detected markers.
   - The frame is displayed using OpenCV's `imshow()`, and the loop continues until the user presses 'q'.
2. **Exit**:

- The loop breaks when 'q' is pressed, and the window is destroyed using `cv2.destroyAllWindows()`.
- `cv2.destroyAllWindows()` is called to clean up and close any open OpenCV windows.

6 Preliminary Tests

Before using the system for real-time tracking, it's important to perform some preliminary tests to ensure everything is working correctly.

6.1 Test 1: Camera Functionality

- Open Thonny or Visual Studio Code and run a simple Python script to capture a single image from the camera.
- Ensure the camera feed is clear and the image appears on the screen.
- If the image is blurry or the camera feed does not appear, check the camera settings and calibration.

6.2 Test 2: ArUco Marker Detection

- Run the marker generation script to create a set of ArUco markers.
- Place the generated markers in the camera's field of view.
- Run the marker pose estimation script and verify that the markers are detected and their position is accurately displayed.
- If the markers are not detected, check the camera alignment and lighting conditions in the environment.

6.3 Test 3: Communication with ESP32

- Ensure that both the Raspberry Pi and the ESP32 are connected to the same Wi-Fi network.
- Run the code that sends position data from the Raspberry Pi to the ESP32.
- Verify that the ESP32 is receiving and processing the data correctly by using serial output or any other debug method.

6.4 AruCo Markers: A Brief Introduction

ArUco markers are square-shaped fiducial markers used in computer vision, particularly for tasks like augmented reality and camera pose estimation. Each marker contains a unique ID encoded in a binary grid, typically consisting of black and white squares. The markers are easily detectable due to their high contrast and distinctive pattern, even when rotated or partially occluded. In OpenCV, ArUco markers are detected using the `cv2.aruco` module, which can identify the marker's position and orientation in an image. The detection process involves converting the image to grayscale, applying an algorithm to locate the marker's corners, and then decoding the ID. OpenCV can also estimate the 3D pose of the marker relative to the camera if the camera calibration data is available, making ArUco markers ideal for tracking and overlaying virtual objects in real-world environments.

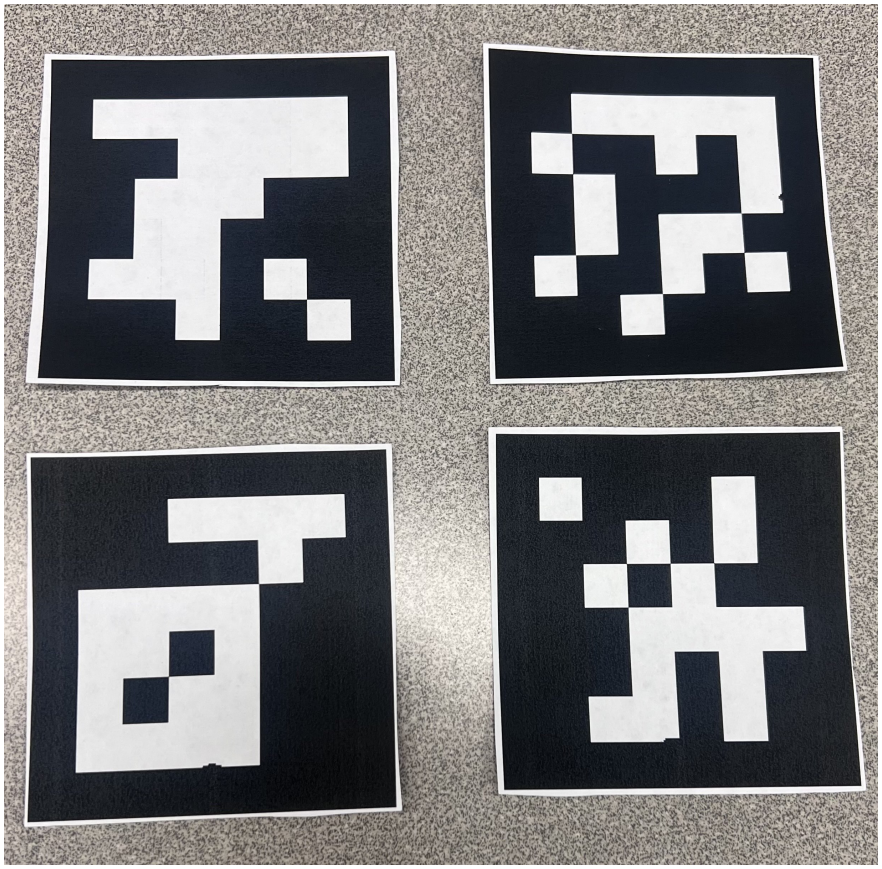

Figure 8: Sample ArUco Markers

7 Conclusion

The Pseudo GPS for Romi system is a robust solution for tracking the positions and orientations of Romi robots within a lab environment. By utilizing a ceiling-mounted camera and ArUco marker detection, this system offers a proof of concept for precise real-time localization of robots.