Leader-Follower Robot Movement Control

Bradley Harris Student Number: 120385201

March 2023

1 Introduction

This project aims to emulate the leader-follower paradigm through developing and merging a variety of vision and movement control algorithms implemented on a group of mobile robots. This is in effort to effectively implement leader-follower movement, with an emphasis on tracking performance, algorithmic simplicity and cost. The Leader leads the Follower either by line following on a track, using a pre-programmed course or following another subject. The following robot then uses its camera to track the position, velocity and course of the leader in an attempt to emulate its path accurately.

This is done using two Jetbots[1] as the agents and using a Raspberry Pi[2] as an access point. As seen in the architectural overview in figure 1, the Jetbots are based on the NVIDIA Jetson Nano[3] single-board computer and feature a simple CSI camera as well as a two wheel differential drive system.

This paper will discuss the discoveries made throughout the process of this project to date.

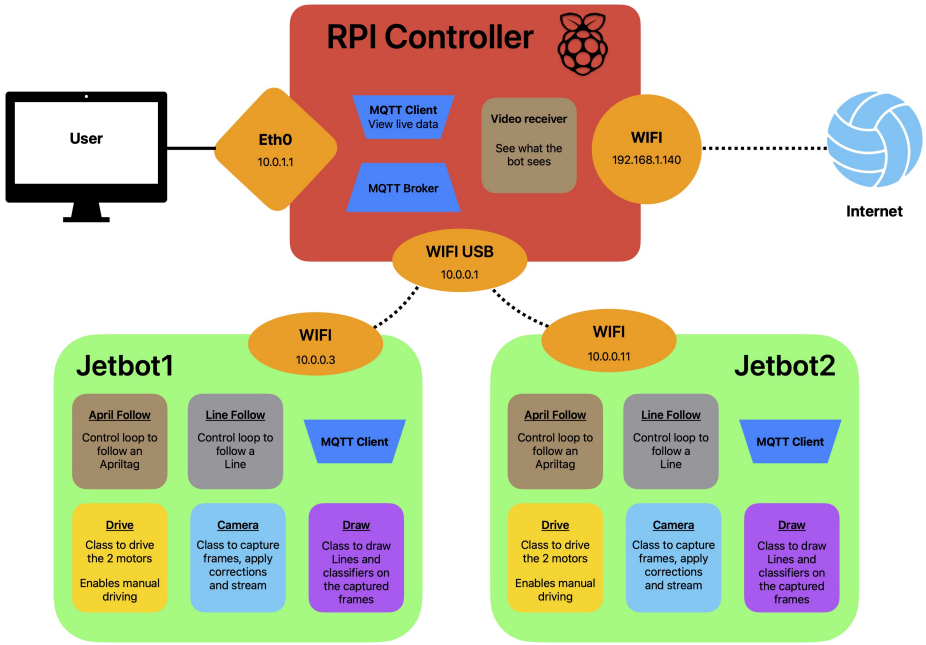

Figure 1: An architectural overview of the system

[1]Jetbot AI Kit, https://www.waveshare.com/wiki/Jetbot_AI_Kit
[2]Raspberry Pi 4, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[3]Jetson Nano, https://developer.nvidia.com/embedded/jetson-nano-developer-kit

2 Findings

2.1 Message Queuing Telemetry Transport (Mqqt) as useful Light and Fast Communication

Obtaining data from the Jetbots in real time is very important for attempting to tune PID loops (See 2.6 PID Controllers) and understand behaviours that may arise due to errors in the implementation. MQTT[4] is ideal for this situation. The broker that runs on the Raspberry Pi, receives the messages from both of the Jetbots. The Broker can then distribute the messages to any programs that want to access the data. This data can be anything from motor values, states, or other information.

MQTT is lightweight and does not require very much bandwidth which is ideal over noisy, unreliable wireless connections. There are compatible libraries[5] in many languages for easy adoption and it is non-blocking which is especially useful for its real-time applications. The individual quality of service levels enable flexibility depending on the requirements of the application. Clients can disconnect from the broker and leave a last will and testament which notifies all subscribers that the channel has been abandoned.

This system was especially helpful when tuning the PID loops for steering as the individual potential, differential and integral component could be compared and their contribution could be evaluated and altered.

To do this effectively graphing the values, like in Figure 4, in real time, is desirable. Grafana is a commonly used, interactive, data visualisation platform with a web based ui, that can be ran locally. It has built in support for a variety of major databases and data types. Its advertised as fast and easy to setup. However MQTT support has been deprecated and exists with an old plugin that will only build with some workarounds. Even after building this plugin does not connect to the broker. Using JSON is not fast enough, and likewise connecting a web-socket to the broker has been deprecated. Attempting to graph live PID values with matplotlib's pyplot gave similar results, the graph could not update fast enough.

2.2 Platform Limitations

The hardware platform has some limitations on how accurately the software could work.

The 8 megapixel IMX219 sensor CSI camera that came in the Jetbot kit. Due to the IMX219 sensor CSI camera and processing power of the Jetson Nano that came in the Jetbot kit, the usable resolution for this project us 720p at 30FPS. The gstreamer[6] interface exposes a lot of configurations for the camera such as exposure time, focus, resolution and frame-rate. However attempting to change the exposure to increase the amount of light that can get into the sensor in an effort to reduce motion blur, resulted in a blurry or static signal. This unusable image caused the Apriltag detection function to not perform very well. As seen in Figure 2 when the Apriltag moves too fast across the screen. Subsequently, the steering algorithm becomes unresponsive and jittery as the subject cannot be consistently tracked from frame to frame.

The camera's FOV is 160°. A larger FOV allows for tracking across a larger area. However the camera has a limited amount of individual rgb sensors, and as the FOV increases, the pixels per degree decreases. This impacts the ability of the detection algorithm on finding Apriltags

[4]MQTT, `https://mqtt.org/`

[5]Python MQTT Library, `https://pypi.org/project/paho-mqtt/`

[6]gstreamer, `https://gstreamer.freedesktop.org/documentation/index.html`

that are far away. Using larger tags is an option as there are many families to choose from of various shapes and sizes, but the smaller ones worked best for this application.

Figure 2: Loss of tracking over 4 frames due to motion blur

The two DC motors used to drive the wheels are controlled via a motor driver that passes PWM signals to each motor. By the very nature of PWM at lower values, the responsiveness of these motors is poor, as seen in Figure 3. This means that they cannot be driven with great resolution at low speeds. This means that the drive system is nonlinear. Choosing to use a PID controller to drive the steering was the best choice, however they do not perform very well if the output is nonlinear. To overcome this, an override was used which relied on the fact that the motors were already spinning due to forward motion and combined the steering values to get a good result.

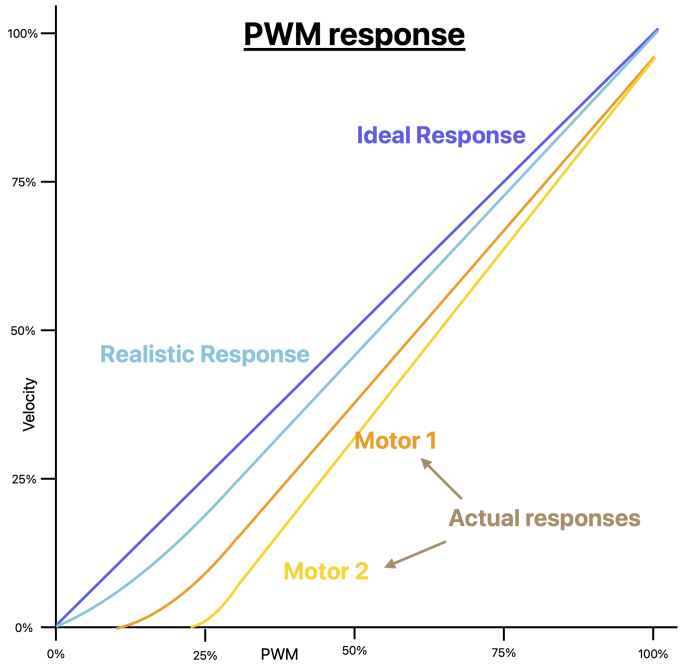

Figure 3: The PWM response of each motor (estimated)

No two mechanical devices are built equal, and the cheap TT motors used here are no exception. Figure 3 shows the estimated PWM response of each motor. They are unbalanced and this leads to drift when driving each motor with the same value. In addition to an override, section 2.6 outlines a way to counter for this.

2.3 Raspberry Pi

To connect this project, make it transportable and to ease development A Raspberry Pi is used as an access point between the agents. Acting as a hub for development and as a gateway to the robot network, it provides ease of use and portability, which is important when developing at university or at home. This Raspberry Pi can run many useful applications such as MQTT brokers, interfaces, services, and offload compute from the agents. In this star topology, the Raspberry Pie acts as a sink to the Jetbot agents. The agents can be accessed via SSH through the Raspberry Pie from the management port (Eth0) as seen in Figure 1

This approach provided a base that enabled supervision and control of the system. Being able to SSH through to each of the Jetbots with their own network provides simplicity and isolation from surrounding networks. Running applications that the agents use, or to combine data from the agents locally.

2.4 Apriltag Versatility

Apriltags[7] are QR code like markers used as targets in machine vision applications. They come in a variety of family types that all convey ID and other attributes in different shapes and sizes. Processing of these markers is lightweight and can be done in real time, taking the same approach as QR codes. They encode 4 to 12 bits usually representing the ID that allow them to be identified more robustly from further away.

The Apriltag library[8] provides a detect function that can be called on an image or frame. Detect provides the decoded ID, the decision margin, and a homography matrix[9]. It provides the centre and corner coordinates within the frame of the tags. By defining the size and camera parameters, detect will also return the rotation, translation, and error of the pose.

Using this information, it's easy to track an agent within a frame and gather useful positional data about the agent or the motion of the camera. The libraries are easily accessible with many exposed features and compatibility with other vision packages such as OpenCV[10].

2.5 Consistency of Data

Working, purely in the code domain without having to touch the physical world is ideal. Reliability, consistency, and structure can all be counted on as constants. However, when dealing within the real world, integers never exist, Reals are the only values. Constructing a code base that is resilient against the real world is very difficult.

The inconsistencies introduced through the images that the camera produces, such as motion blur, grain, or exposure, can cause errors and hallucinations within the detection algorithms. PID controllers rely heavily on the consistency of data and if ther is an absent or rogue point it can throw off the differential or the integral very fast. This can introduce oscillations or complete deviation to the system. The detector, imagining tags, or losing tracking of a real time can be catastrophic if the correct protections are not implemented. Relying on a more consistent detector like a light Follower or colour Follower could be more consistent, but the same abundance of data cannot be derived from those markers.

[7] Apriltags, `https://april.eecs.umich.edu/software/apriltag`

[8] Python Apriltag Library, `https://github.com/duckietown/lib-dt-apriltags`

[9] Homography, `https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html`

[10] OpenCV, `https://docs.opencv.org/4.x/`

2.6 PID Controllers

A PID controller is a prolific way of steering a system towards the desired target, value, or level. Proportional is the difference between the desired value and the current value, Integral is the error over time, and Differential is the change in error between the last iteration and the current.

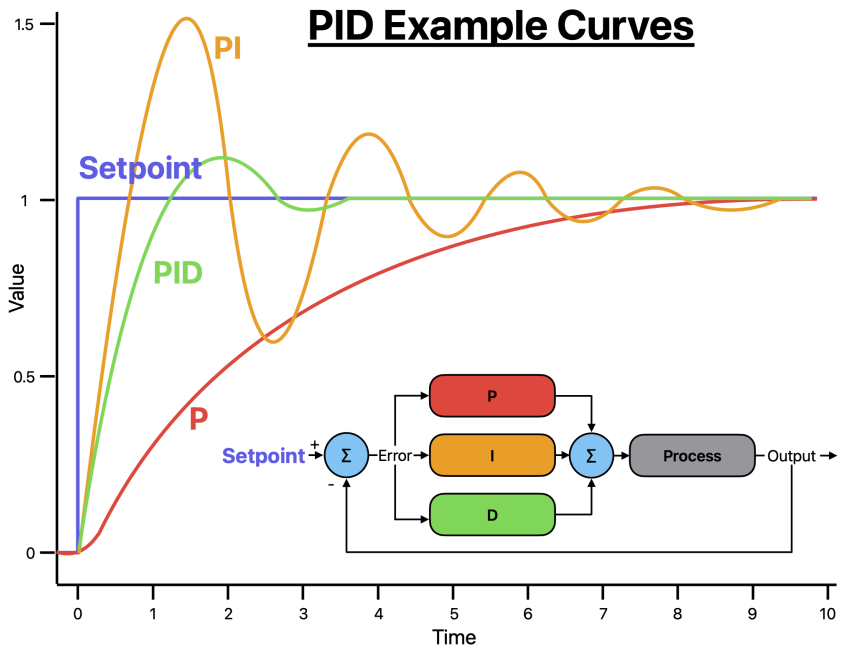

Figure 4: How the individual PID values influence the system

PID controls are ideal for Systems with a linear output. They can deal with quadratic inputs, but struggle with the complexities above that. If the system requires a nonlinear output, additional wrappers or more advanced control techniques may be required.

For this system it could be advantageous to implement individual PID controllers for each of the motors. This would create a linear interface for the nonlinear response of the motors, as outlined above. However to achieve this we need feedback from the motors using encoders.

3 Conclusion

The findings within this document outline the issues and challenges faced when implementing a leader follower robot system. The architecture created provided smooth, fast development across multiple systems, in any location. MQTT Enables lightweight asynchronous communication between the nodes, which was ideal for tuning and monitoring of the system. Limitations in the hardware were identified in the forms of non-perfect drive systems and camera. The motion blur and sensor issues are commonly faced when creating machine vision applications. Apriltags are easy to use and provide targets in the real world that can be tracked. They can be used to easily translate motion in the pixel domain to real world units. However the consistency of tracking data can cause issues for the PID controllers and the detector can be prone to hallucinations. PID controllers are ideal for control smoothing with tune-able parameters to

control the reaction of system. Tuning the parameters was found to be challenging given the platform inconsistencies.

To conclude, when implementing a leader follower robot system a reliable hardware platform will ensure ease of use and reduce the amount of software compensation required. Setting up the architecture correctly early on creates a salable environment that more agents can be added to later on. Recommendations for further investigations include: Porting the software to a more predictable platform, Distributed architecture for fine control of wheel speed, improved motion blur compensation, and utilising the pose provided by the Apriltag to predict motion vectors.