Final Project — AU Robotics

Preface

This report serves as a technical overview of everything that has been worked on in the past two weeks by Team B in the final project for AU Robotics in 2024.
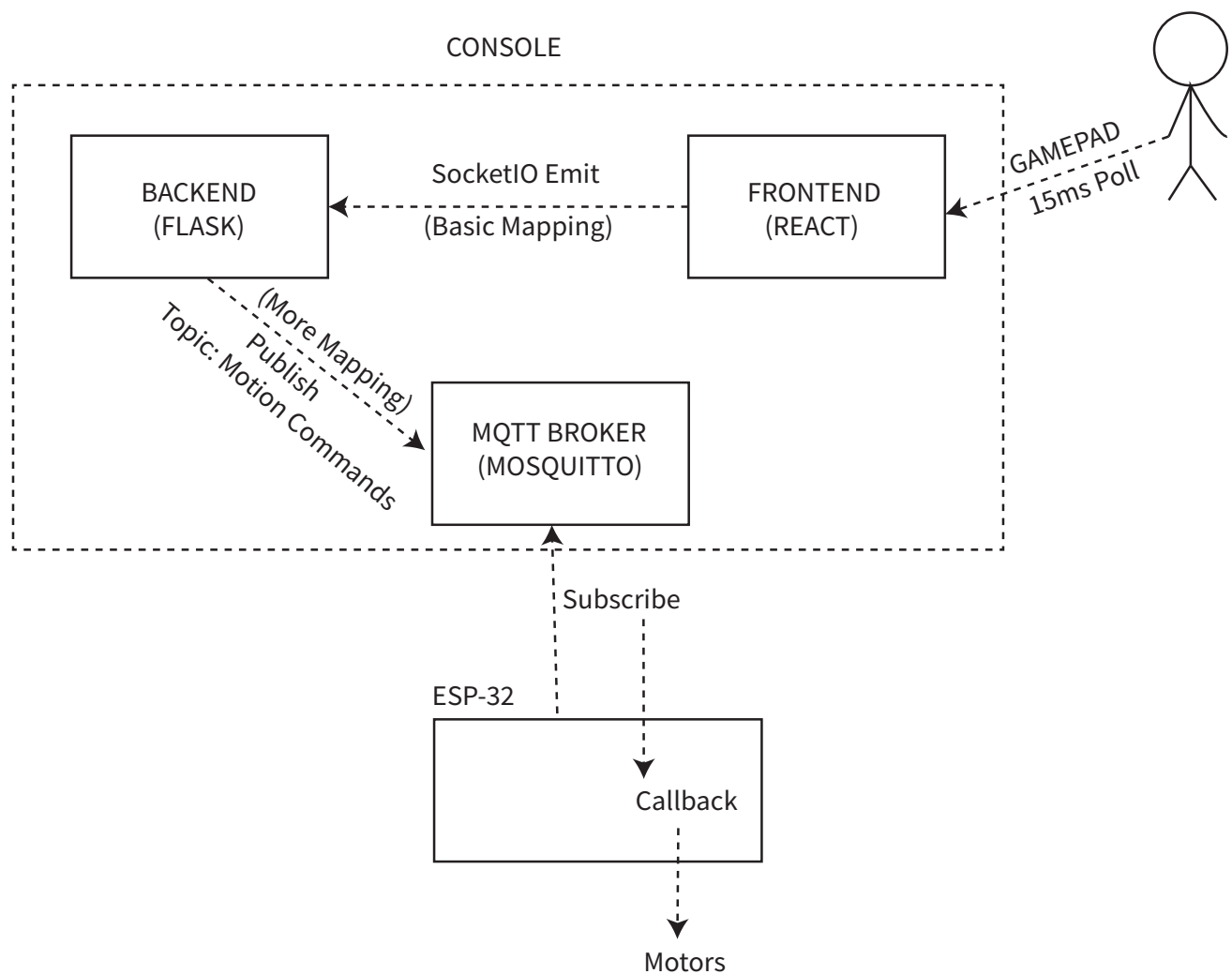
Contents

Part I

Software

1 Overview & Dataflow

1.1 Motion Dataflow

CONSOLE

BACKEND
(FLASK)

SocketIO Emit
(Basic Mapping)

FRONTEND
(REACT)

GAMEPAD
15ms Poll

(More Mapping)
Publish
Topic: Motion Commands

MQTT BROKER
(MOSQUITTO)

Subscribe

ESP-32

Callback

Motors

1.2   Localization Dataflow

CONSOLE

BACKEND
(FLASK)

SocketIO Emit

Coordinates
(Result from Fusion)

FRONTEND
(REACT)

Subscribe

Callback

MQTT BROKER
(MOSQUITTO)

Linear
Accel.
+Gyro

HTTP GET

Mobile

Publish

Topic: Encoders

IMU
(Phyphox)

ESP-32

30ms pass

0.25ms Push

Encoders
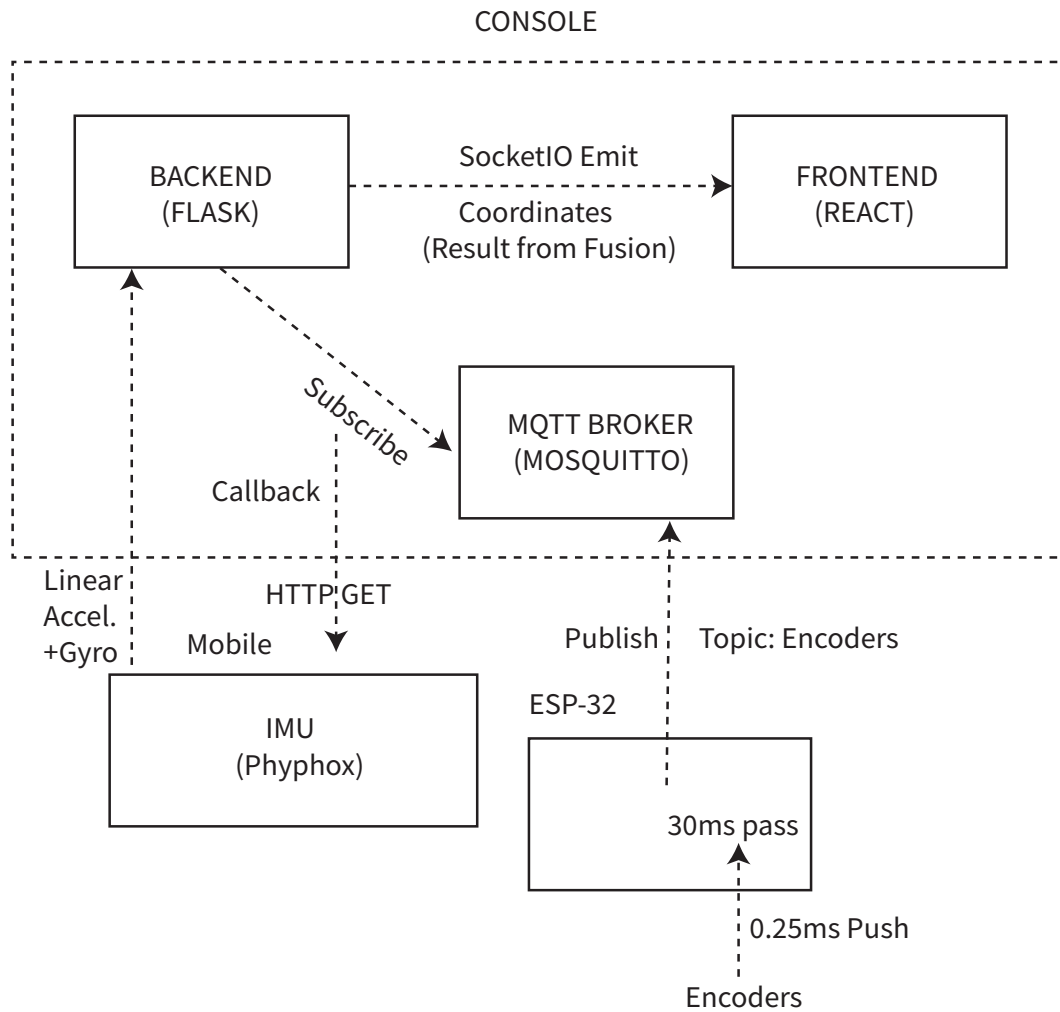
1.3   Computer Vision Dataflow

CONSOLE

BACKEND
(FLASK)

Video Feed, SocketIO Emit

Numbered Boxes
With Coordinates

FRONTEND
(REACT)

Virtual
Camera Device

HTTP Connection

Mobile

Camera
(Droidcam)
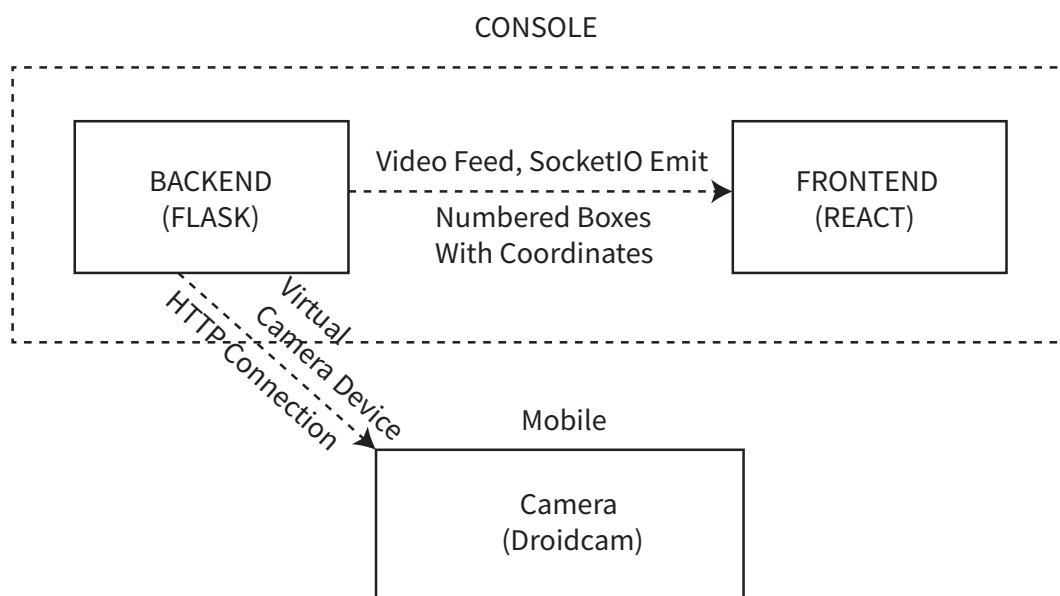
2 Embedded

Goals

Programming the `ESP-32WROOM` to do the following:

- receive motion commands from the console and translating them into something the motors can understand

- translate commands related to the arm and gripper into something the servo can understand

- send encoder data to the console to enable it to carry out its localization routine

3 Localization & Motion

Goals

With the power of the laptop used for the *Console*, the localization team should efficiently and accurately calculate the coordinates.

The team will receive the encoder readings from the `ESP-32WROOM` as shown in Section 1.2. There is little communication overhead with the `ESP-32WROOM` due to the fact that the `MQTT Broker` is running locally on the *Console*. After receiving the encoder data, the IMU readings — which are the linear acceleration in the x direction, and the angular acceleration in the z direction — are requested from the mobile phone mounted on the robot.

A Kalman filter will be performed on the accelerations to obtain their respective velocities, then fusion of all the data will be performed to calculate the current coordinates and dispatch them to the frontend.

4 Console

The *Console* is the powerhouse that all things computable get pushed to. As an extension, it also orchestrates much of the communication between the devices and processes the video feed for detection and decoding of QR codes.

All this is an aside to its main function: allowing the user to control the robot, navigate the map, and track objectives. The *Console* contains the following UI components:

(1) A video feed display, with buttons to set the vision algorithm

(2) A dynamic list of all boxes and their respective coordinates

(3) A dynamic map being updated with the robot's position

4.1 Navigation & Control

Using the Gamepad API[1] which is standard in all mainstream web browsers, we are able to process gamepad button presses in a game-loop–like algorithm which polls the gamepad every 15ms.

The algorithm assigns *precedence* to certain inputs, so for example, if the user moves the analog stick used for rotation and also presses the trigger used for forward motion, the algorithm only sends the rotation movement. This is the command format along with the precedence for each command:

Index	DS* Button	Command	Value
0	×	Arm Down	0
0	△	Arm Up	1
1	○	Grip Close	0
1	□	Grip Open	1
2	R2/ L2	Both Wheels ↑/ ↓	-1 → 1
3	R Stick →/ ←	One Wheel (Rotation)	-1 → 0 : left 0 → 1 : right

* DualShock ® or DualSense ® which belong to Sony ™

The reason for using this precedence is to avoid damaging the wheels' differential, by allowing only one type of motion to go through.

As clarification for the forward and backward motion, the command's value is calculated by subtracting the R2 value form the L2 value and sending the command accordingly.

The right analog stick is calibrated using the controller's firmware by doing a full rotation before running the robot, and a further error of 0.1 in any direction is accounted for — the vertical directions are disregarded as are all other inputs.

---

[1] https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API

4.2 Computer Vision

The role of computer vision is to detect and decode the QR codes on every box and assign a box number to each code, allowing the individual in charge of navigation to effectively track which boxes need to go where.

The algorithms that are made available:

Library	Strengths	Weaknesses
pyzbar	Extremely Efficient	Angled Codes
cv2 - builtin	Angled Codes	Less Efficient
qreader	Deepest Detection	Least Efficient

The individual in charge of navigation will be able to switch between the algorithms on-the-fly or even disable them completely. This would also be useful in cases of autonomous or semi-autonomous navigation.

4.3 Communication

Several of the communication components are hosted directly, either partly or fully, on the *Console*.

Host	Component	Connects to
Mobile	IMU	*Console* w/ HTTP GET
Mobile	Camera	*Console* w/ HTTP
Console	MQTT Broker	ESP-32WROOM
Console – Backend	Flask & SocketIO	Frontend

The full dataflow is illustrated in Section 1.

5 Setup Instructions

Setup instructions for every software component will be indicated and updated in the repositories various README files.