Final Project — AU Robotics

29/10/2024 **Team B**

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.

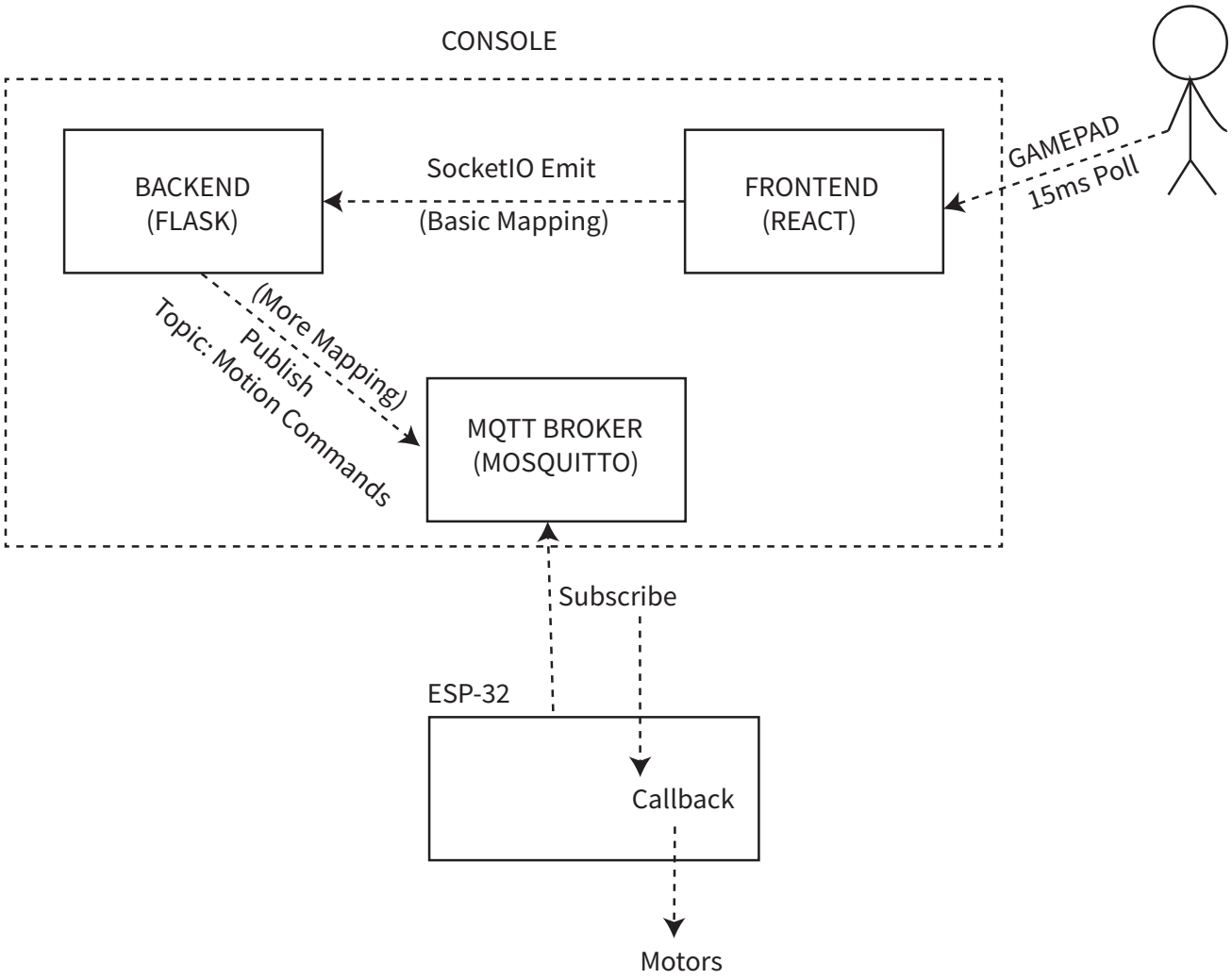
Contents

Part I

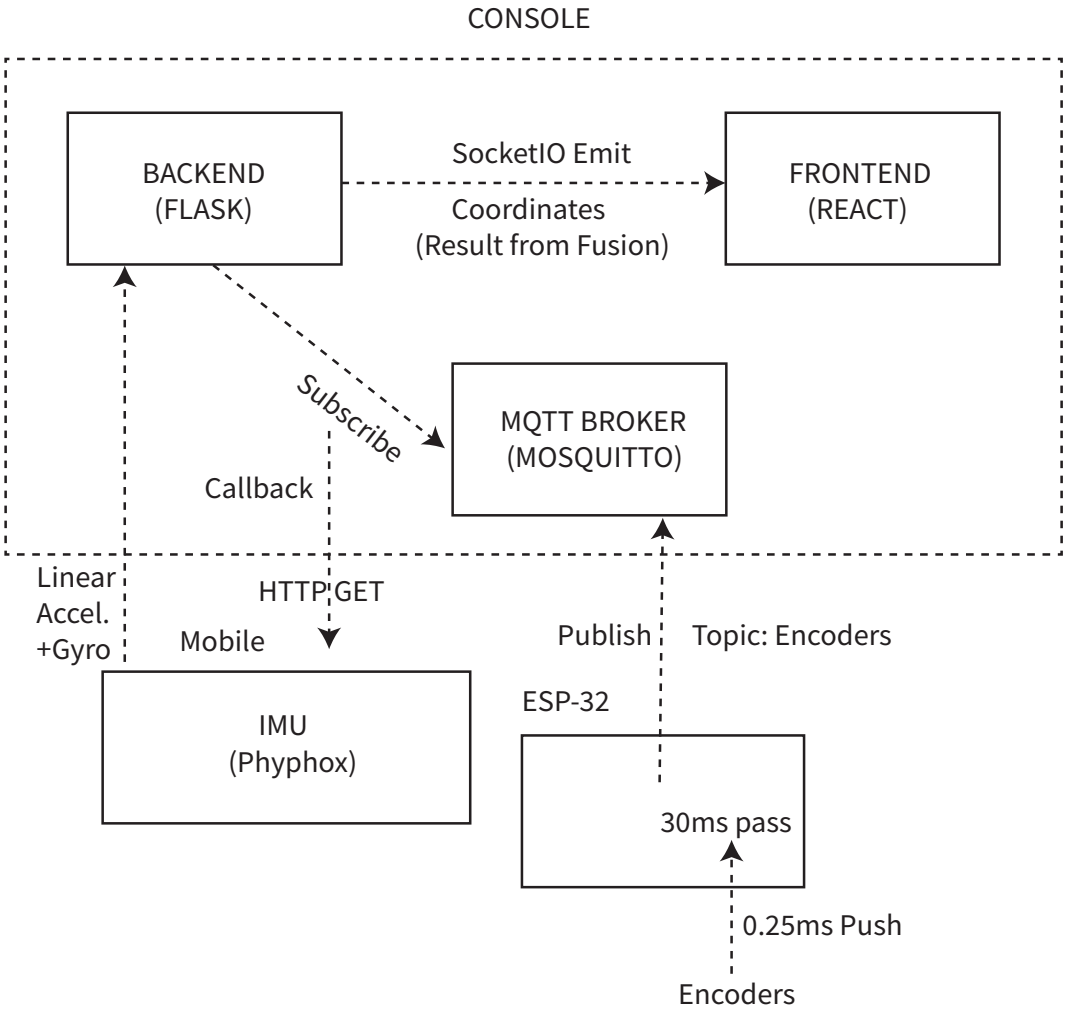
Software

1 Overview & Dataflow

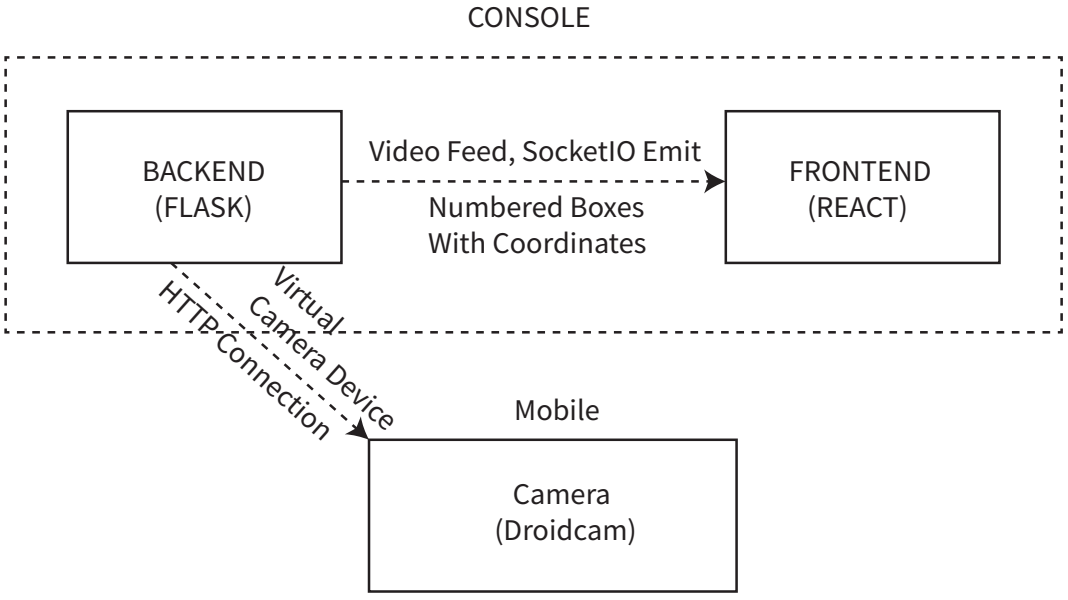
1.1 Motion Dataflow



1.2 Localization Dataflow



1.3 Computer Vision Dataflow



2 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

2.1 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. Precedence is simulated using a switch–like algorithm with early exits.
This is the command format along with the precedence for each command:

Index	**DS* Button**	**Command**	**Value**
0	$\times$	Arm Down	0
0	$\triangle$	Arm Up	1
1	$\bigcirc$	Grip Close	0
1	$\square$	Grip Open	1
2	R2/ L2	Both Wheels $\uparrow$/ $\downarrow$	-1 $\rightarrow$ 1
3	R Stick $\rightarrow$/ $\leftarrow$	One Wheel (Rotation)	-1 $\rightarrow$ 0 : left 0 $\rightarrow$ 1 : right

* DualShock ® or DualSense ® which belong to Sony ™

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

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 L2 value form the R2 value and sending the command accordingly.

The right analog stick is calibrated using the controller's firmware by doing a full rotation while connected to the *Console* 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.

2.2 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.

2.3 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

Mutlithreading for the different event handlers is handled graciously by the libraries used (`Flask`, `Flask-SocketIO`, `Paho-MQTT`). As an example, `Paho-MQTT`'s `loop_start` and `loop_stop` are used instead of the thread-blocking `loop_forever` which requires manual thread handling.

The full dataflow is illustrated in Section 1.

3 Embedded

Goals

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

- receive motion commands from the console and translate 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

4 Localization & Motion

With the power of the laptop used for the *Console*, the localization team should efficiently and accurately calculate the coordinates and send correct commands from the gamepad to the `ESP-32WROOM`.

Goals

- Map $0 \rightarrow 1$ values to $0 \rightarrow 255$ values
- Prepare values for differential steering (i.e if command is to rotate to the right with a speed x, add a speed of x in the right wheel and a speed of -x in the left wheel)
- Receive encoder readings from `ESP-32WROOM`and IMU readings from mobile phone
- Apply a Kalman filter and perform sensor fusion to calculate current coordinates

4.1 Input Interpolation & Dispatching

The navigation and motion team is responsible for translating the table in 2.1 to the following table in a way that conforms with the first two goals. The team has chosen to use numpy to perform linear interpolation on the code, as an example:

```
from . import mqttclient
publish = mqttclient.publish
@socketio.on("gamepad buttons")
def map_and_send_buttons(command):
    # ex: command == "3,0.7612"
    index = int(command[0])
    value = float(command[2:])
    # ...
    elif index == 3:  # Rotate left or right
    if value > 0:
        # Rotate right
        right = (
            "2,"
            + str(int(np.interp(value, [0, 1], [0, 255])))
            + ","
            + str(int(np.interp(value, [0, 1], [0, -255])))
        )
        publish("Motion Commands", right)
    elif value < 0:
        # Rotate left
        left = (
            "2,"
            + str(int(np.interp(value, [0, 1], [0, -255])))
            + ","
            + str(int(np.interp(value, [0, 1], [0, 255])))
        )
        publish("Motion Commands", left)
    else:
        print("Unknown index received")
```

This choice was made due to the ease of using numpy and the lower risk of human error.

The `mqtt` client object is passed from the backend directly to the function call, which is run on its own thread that is managed by the `Paho-MQTT`. It can publish directly without re-establishing the connection or closing it after publish. Establishing the connection, reconnecting on disconnect,

and keeping it alive is managed by `Paho-MQTT` as well.

4.2 Extended Kalman Filter

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.

```
imu_measurement = np.array([[aX], [0], [gZ]])
ekf.process_measurement(timestamp, "IMU", imu_measurement)
wheel_measurement = np.array([[eL], [eR], [0]])
ekf.process_measurement(timestamp, "WHEEL_ENCODER", wheel_measurement)
# ... ekf called, see snippet below
```

As soon as the robot starts motion, the state vector is initialized to zero vector to indicate the zero coordinates and the `is_initialized` variable is set to `False` to indicate that the next time data is sent the coordinates aren't reset back to zero.

```
self.ekf_.predict()
if sensor_type == "IMU":
    self.ekf_.update_imu(measurement)
elif sensor_type == "WHEEL_ENCODER":
    # measurement: [eL, eR, 0] where eL and eR are the distances from the left
    and right encoders
    dL = measurement[0, 0]  # Distance from left encoder
    dR = measurement[1, 0]  # Distance from right encoder
    # Calculate delta values
    delta_d = (dL + dR) / 2.0  # Average distance
    delta_yaw = (dR - dL) / self.L  # Change in yaw
```

After processing the IMU and encoder data inside the `process_measurement` function, we predict the robot's position based on the IMU data, the state vector is updated with the x, y positions and yaw from IMU, then we update the robot's position based on the data from encoders, this predicted position is compared with the position obtained from encoders.

The Kalman filter uses this to fuse the IMU and encoders.

5 Setup Instructions

Setup instructions for every software component will be indicated and updated in the repository's various README files. The repository will be hosted remotely at `https://github.com/y-samy/aur-fp24`

Part II

Hardware

6 Mechanical

6.1 Motors

$F_T = F_{roll} + F_{incline} + F_{accelerate}$
$F_{roll} = \text{Weight} \times \text{Crr}$
$F_{incline} = Weight \times \sin(\theta)$

Assumptions

Mass = 6kg and $\theta = 0$ (no inclined surface in playground), and $a = 0.5 m/s^2$
Efficiency 70%
Then the output torque of the motor is $\approx$ 0.9kg.cm
RPM = 100

High torque and low RPM `DC Motor GA25-370` was chosen to reach acceleration rather than high top speeds, as the playground is relatively small.

The motors are equiped with encoders to enable localization.

Two servos were chosen, `MG945` with 10.5 kg.cm torque, `SG5010` with 5.5 torque. The former is assigned to raising and lowering the arm, and the latter opens and closes the gripper that holds the boxes.

6.2 Wheels

`Robot Wheel 65mm` was chosen as it has high grip rubber, and its diameter enables even more grip to increase acceleration.

Wheel	Acc.	Rotation	Cost
`Freewheel`	Worst	Best	Cheapest
`Bearing Wheel`	Best	Bad	More Expensive
`Wheel Belt`	Good	Good	Most Expensive

6.3 Materials & Dimensions

Wood was chosen for the chassis over sheet metal and aluminum as it is lighter, easier to modify and shape, and is a lot cheaper. The wood was cut using *laser cutting*.

The dimensions are 400 $\times$ 226 mm, the wheels are not included in the width.

7 Electrical

`L293D` Motor Driver IC

7.1 Overview

The `L293D` is a dual H-bridge motor driver integrated circuit (IC) widely used for driving small DC motors and stepper motors in both directions. This IC is designed to control the direction and speed of two motors simultaneously, making it ideal for robotics and other projects requiring precise motor control.

The `L293D` can supply a motor with up to **600mA of continuous current** and **36V**. It includes **internal diodes** for back-EMF protection, making it robust and durable when dealing with inductive loads like motors. This IC uses TTL-compatible logic inputs and requires a dual power supply: one for the motor power (Vcc2, up to 36V) and another for the logic control (Vcc1, typically 5V).

7.2 Features

- **Dual H-bridge**: Controls two motors independently
- **Operating voltage**: Logic voltage of 4.5V to 7V (Vcc1) and motor voltage of up to 36V (Vcc2)

- **Current Handling**: Up to 600mA continuous current per channel (1.2A peak)
- **Thermal shutdown**: Protects the IC from overheating
- **Internal protection diodes**: Protect against back-EMF generated by motors
- **Bidirectional control**: Supports forward and reverse motion
- **Enable pins**: Allow control of motor states (enabled/disabled)

7.3 Pinout & Functionality

7.3.1 Pinout

The `L293D` has **16 pins**:

1. Vcc1 (Pin 1): Logic voltage supply (4.5V - 7V)
2. Enable 1,2 (Pin 1): Controls the operation of Motor 1. HIGH to enable
3. Input 1 (Pin 2) and Input 2 (Pin 7): Control the direction of Motor 1. Used with Enable 1,2
4. Output 1 (Pin 3) and Output 2 (Pin 6): Connected to Motor 1
5. Ground (Pins 4, 5, 12, and 13): Common ground for the circuit
6. Vcc2 (Pin 8): Motor voltage supply (up to 36V)
7. Enable 3,4 (Pin 9): Controls the operation of Motor 2. HIGH to enable
8. Input 3 (Pin 10) and Input 4 (Pin 15): Control the direction of Motor 2
9. Output 3 (Pin 11) and Output 4 (Pin 14): Connected to Motor 2

Each pair of inputs and outputs represents one H-bridge, meaning two motors can be controlled independently.

7.3.2 Working Principle

The L293D operates as a **dual H-bridge** motor driver, where each H-bridge can drive one motor:

1. **Enable Pins** (1, 9): Enable or disable the motors.
2. **Input Pins** (2, 7, 10, 15): These inputs determine the direction of the motors.
   - When **Input 1** is HIGH and **Input 2** is LOW, **Output 1** will be HIGH and **Output 2** will be LOW, causing the motor to rotate in one direction.
   - Reversing the inputs (Input 1 LOW, Input 2 HIGH) will reverse the motor direction.
3. **Power Pins** (Vcc1, Vcc2): Provide the necessary power for the motor and logic functions. Vcc1 is the logic voltage, and Vcc2 is the motor supply voltage.
4. **Output Pins** (3, 6, 11, 14): Provide the driven output to the motor.

The L293D's **internal protection diodes** ensure that any back-EMF generated by the motors does not damage the IC, which is especially useful for inductive loads.

7.4 Difference Between L293D and L293

Both the L293D and L293 are dual H-bridge motor drivers, but they have a few key differences:

1. **Back-EMF Protection**:
   - L293D includes **internal protection diodes** for back-EMF, making it safer to use with inductive loads (like motors) without needing external diodes
   - L293 lacks these protection diodes, so external diodes are necessary to prevent back-EMF from damaging the circuit
2. **Current Handling**:
   - L293D is rated for **600mA continuous current per channel** (1.2A peak)

- `L293` can handle a higher **continuous current of 1A per channel** and up to **2A peak**, making it suitable for heavier loads than the `L293D`

3. **Cost and Availability**:
   - The `L293D` is more commonly used in educational and hobbyist projects due to its integrated protection diodes and slightly lower current capacity
   - `L293` may be used in projects requiring higher current without needing internal protection.

7.5 Practical Applications

- **DC Motor Control**: Control small DC motors in robots or automated systems
- **Stepper Motor Driver**: Used in stepper motor applications by switching between coils
- **Robotics and Automation**: Ideal for robot vehicles, automation systems, and servo control

7.6 Circuit Example

(Connecting to a DC Motor)

Below is an example setup to control a single DC motor with the `L293D`:

1. Vcc1 (Pin 16): Connect to 5V
2. Vcc2 (Pin 8): Connect to the motor's supply voltage (e.g., 12V)
3. Ground (Pins 4, 5, 12, 13): Connect to the ground of the power supply
4. Enable 1,2 (Pin 1): Connect to a PWM-enabled pin on the microcontroller to control speed or directly to 5V to enable
5. Input 1 (Pin 2), Input 2 (Pin 7): Connect to microcontroller pins to control motor direction
6. Output 1 (Pin 3), Output 2 (Pin 6): Connect to the motor terminals

7.7 Conclusion

The L293D is a versatile and robust motor driver IC, particularly suited for low-to-medium-power motor control applications due to its internal protection features and ability to handle bidirectional motor control. For applications requiring higher current, the L293 can be used, with additional circuitry for protection against back-EMF. The selection between L293D and L293 depends on the motor's current requirements and the need for internal protection.

7.8 Our Circuit