Task Diagram

ECE 4534 - Team 12

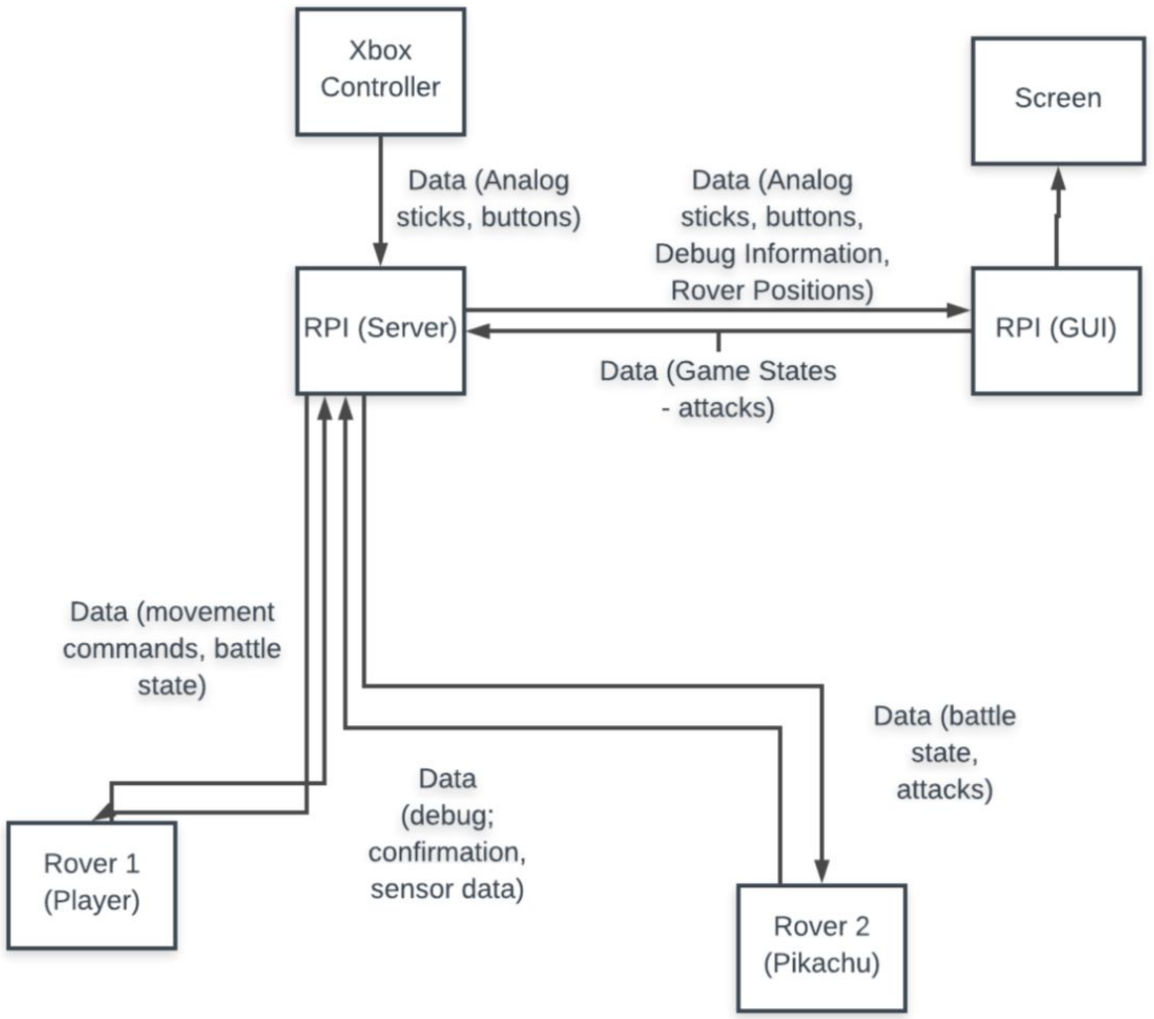
Xbox Controller
Screen
Data (Analog sticks, buttons)
Data (Analog sticks, buttons, Debug Information, Rover Positions)
RPI (Server)
RPI (GUI)
Data (Game States - attacks)
Data (movement commands, battle state)
Data (battle state, attacks)
Data (debug; confirmation, sensor data)
Rover 1 (Player)
Rover 2 (Pikachu)

Figure 1: High Level Communication Diagram

Figure 1 illustrates the high level communication taking place for Pokemon Rover. Every rectangle depicts a physical piece of hardware, where every arrow depicts high level communication taking place. Our low level task diagrams are below.

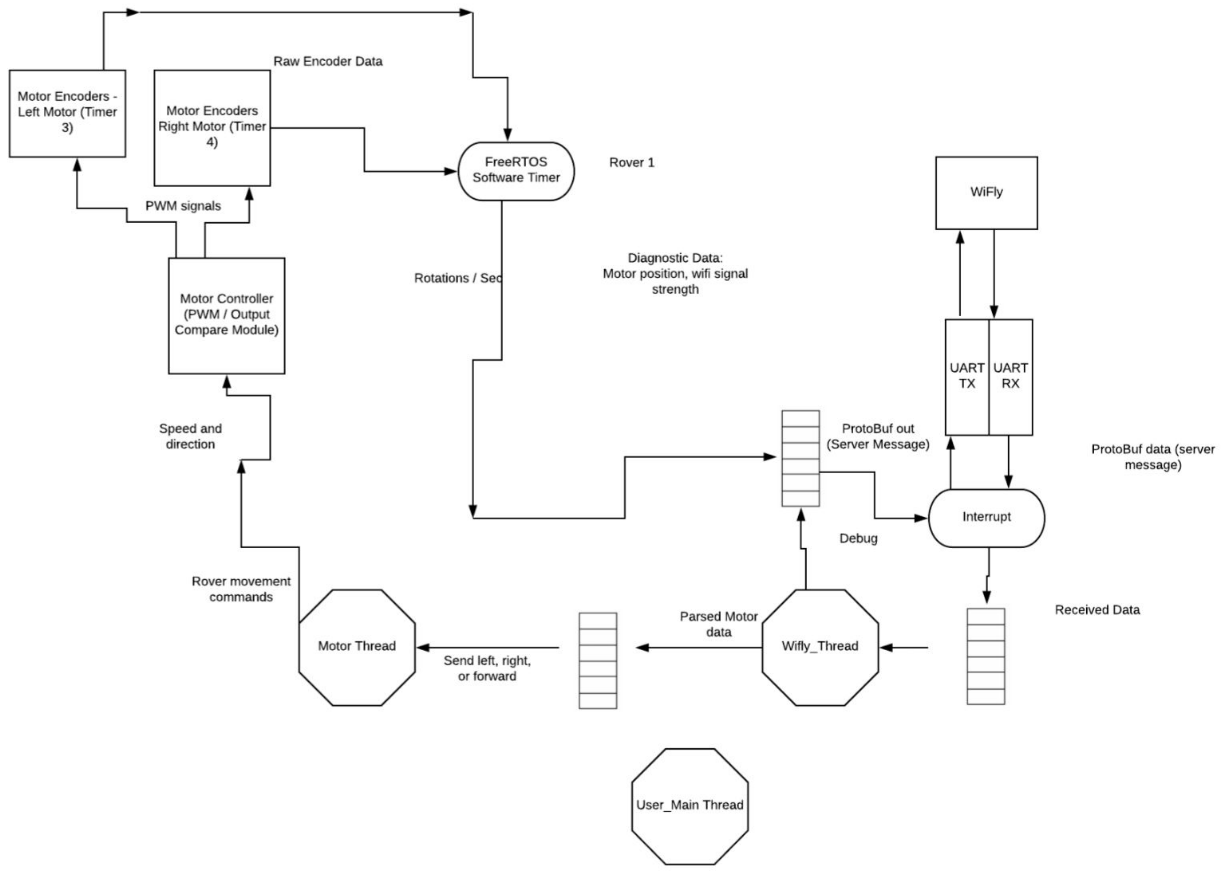
Raw Encoder Data
Motor Encoders - Left Motor (Timer 3)
Motor Encoders Right Motor (Timer 4)
FreeRTOS Software Timer
Rover 1
WiFly
PWM signals
Diagnostic Data: Motor position, wifi signal strength
Rotations / Sec
Motor Controller (PWM / Output Compare Module)
UART TX
UART RX
Speed and direction
ProtoBuf out (Server Message)
ProtoBuf data (server message)
Interrupt
Debug
Rover movement commands
Received Data
Parsed Motor data
Motor Thread
Send left, right, or forward
Wifly_Thread
User_Main Thread

Figure 2: Rover 1 Diagram

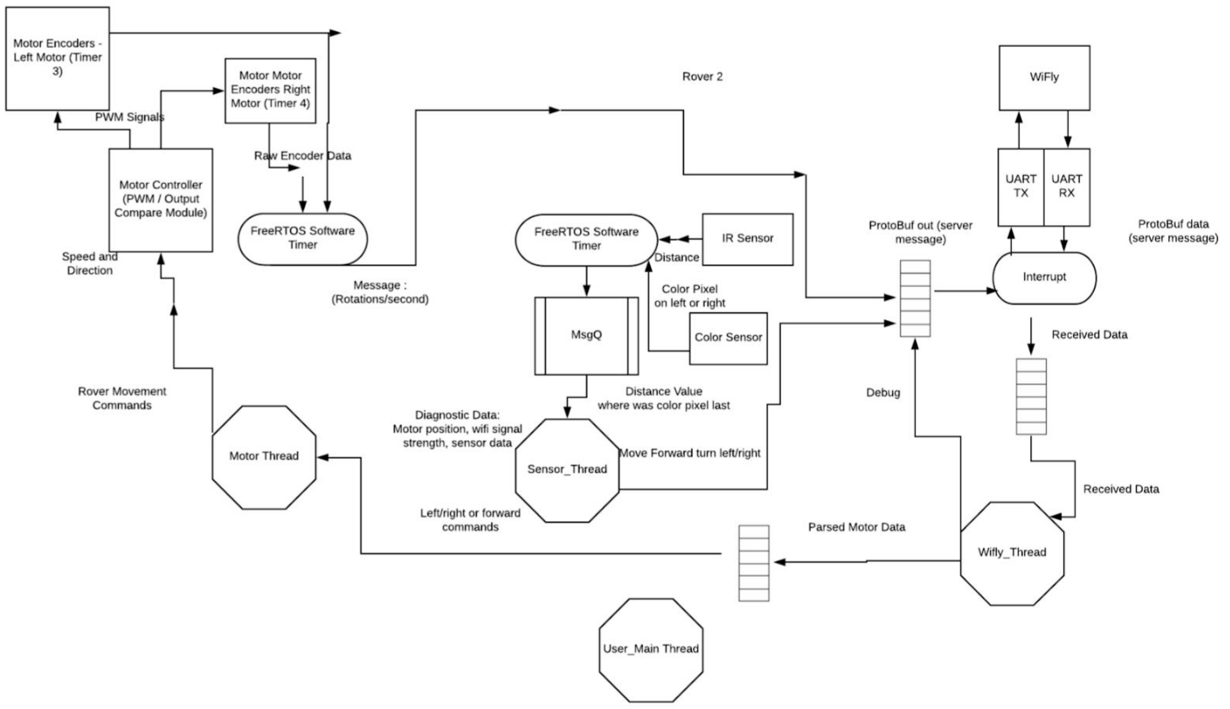
Motor Encoders - Left Motor (Timer 3)
Rover 2
WiFly
Motor Motor Encoders Right Motor (Timer 4)
PWM Signals
Raw Encoder Data
Motor Controller (PWM / Output Compare Module)
UART TX
UART RX
FreeRTOS Software Timer
FreeRTOS Software Timer
IR Sensor
ProtoBuf out (server message)
ProtoBuf data (server message)
Speed and Direction
Distance
Interrupt
Message : (Rotations/second)
Color Pixel on left or right
MsgQ
Color Sensor
Received Data
Rover Movement Commands
Distance Value where was color pixel last
Debug
Diagnostic Data: Motor position, wifi signal strength, sensor data
Motor Thread
Sensor_Thread
Move Forward turn left/right
Received Data
Left/right or forward commands
Parsed Motor Data
Wifly_Thread
User_Main Thread

Figure 3: Rover 2 Diagram

Since the task diagrams are fairly similar for the different rovers, all the information below can be assumed to apply to all the rovers except where specified.

Threads:
User Main Thread: Instantiates and starts all other threads for the program.

Sensor_Thread: The sensor data is retrieved from a message queue filled by a timer based interrupt running in the main thread. Conversion from raw data to actual distance (cm) will be made in this thread. Also for rover 2, the thread should process the data from the color sensor, so that it should notify the motor thread which direction the rover 1 (trainer) has turned. The IR sensor is connected via the ADC built in the PIC32 via connections AIN1, VCC, and GND. Additionally color sensor is connected to VCC,GND and communication of data between the board and sensor will be made via I2C (SDA, SCL). I2C is a simple and flexible way to communicate between two different physically-separated hardwares or contained on the same printed circuit board.

Motor Thread: A message is created that based on the output values read from the motor encoder data from each motor. The message that is created in the ISR contains the encoder data from both motors in the units of rotations per second. From the Motor ISR, 2 subroutines will be called (1 Left, 1 Right Motor) in the Motor Thread. The purpose of these subroutines are to set the speed and direction of the motor based on the motor encoder data message.

Wifly_Thread: Parses json data and sends parsed messages to the correct message queues.

Interrupt Handlers:
Every 125ms, the timer based interrupt should be flagged and sample data from IR sensor and the color sensor. Also, the interrupts should occur for the following events: Data is sent by the server (WiFly), After motor thread has finished processing motor data.

Physical Devices & Interfaces:
Server Raspberry Pi- There will be one server used for communication for all of the physical components of the project. The server will receive movement commands from

the XBox controller, game states from the GUI RPi. Rover 1 will post sensor data and along with rover 2 and 3, rover 1 will send debug confirmations. Rover 4 will receive updates from the emulator. Rover 4 will receive a map from the server, from which the Rover will then take that data and use the A* path finding algorithm to find the nearest exit when the game is over.

GUI Raspberry Pi - Another raspberry pi will be used for emulation, as this is a CPU heavy task. It will be in charge of sending battle states to the server pi, which will then send relevant data to the rovers to get them moving in the game world. Battle commands will also be present and will change the way the rovers act.

IR Sensor - Used for keeping distance between rover 1 and rover 2. Also for detecting obstacles for other rovers.

Color Sensor - The color sensor will be a pixie cam that will read the colors on the rover it is following. It will read different colored sticky notes, representing different sides of the rover it is following. This has been previously used by ECE 4534 teams.

Timer - Timer with interrupt as per documentation; should be set to 125ms.

Message Queues:

The message queues in the rover task diagrams will be able to store a different kind of message. Each WiFly and UART module will send and receive server messages. The different messages for each rover will be encapsulated in the server message. Some examples of the message formats are listed below.

```
Server Message:{
        Int Sequence Number: ,
        Int Message Size (bytes): ,
        String To:                    ,
        String From:                  ,
        String Message Type:    ,
        Oneof Payload: {
                Sensor.sensor sCmd,
                Motor.motor mCmd,
                Emulator.emulation bCmd,
                Status.status stats;
        }
}

Status status:{
        String uid,
        Bool op_complete,
}

Example Payloads:

To_Motor Message:{
        Uint8_t Direction:                ,
        Uint8_t Speed (0-100):   ,
}

From_Motor Message:{
        Encoder_Data: {
                Right Ticks:      ,
                Left Ticks:       ,
        },
}

Motor_Cmd {
        Oneof motor_cmd {
                To_Motor ctl_msg;
                From_Motor data;
        }
}

IR Message:{
        Unsigned int Distance (4- 30cm):
}

Color_Sensor Message: {
        Color_Code:      ,
        Coordinates: [
                Int X:    ,
                Int Y:    ,
        ]
}

Sensor_Msg {
        Oneof sensor_msg {
                IR Message dist_data;
                Color_Sensor color_data;
        }
}

Battle_StartStop Message:{
        Bool PlayerStatus [Win/Loss]: ,
        Bool match_inProgress: ,
}


Battle_State Message:{
        Uint8_t Trainer[Player/Rival]:      ,
        Bool Hit/Miss:    ,
}

Emulation Message:{
        Battle_StartStop battle_ctl,
        Battle_State battle_move;
}

PathFinder Message:{
        int MapData:  [],
}
Debug Message:{
        String Id:
        Bool OperationComplete:
}

From_Emulator:{
        Bool confirm
}
XBox Controller Message:{
        Uint8_t Dpad:
        Bool aButton:
        Bool bButton:
        Bool start:
        Bool reset:
}
```

References

https://www.youtube.com/watch?v=jbPZe6KZe5I&feature=youtu.be: skip to 5:40 for follow rover
https://www.youtube.com/watch?v=HCUwkuRkQ_A&t=1s skip to 7:02 for motor tasks

https://www.youtube.com/watch?v=RfEC0qN8J_Y&t=617s Format of messages