Software design description – Hardware

System overview:

The Mower is a vehicle constructed with an Arduino Mega2560 and a Raspberry Pi zero w. The Arduino controls the mower's sensors and motors, while the Raspberry Pi handle communication between the mower, Frontend, and Backend.

The Mower has two separate modes, manual and autonomous. When in manual mode the mower takes commands from a device connected with Bluetooth. When in autonomous mode the mower contains itself within a specified area while also avoiding front-end collisions.

System architecture:

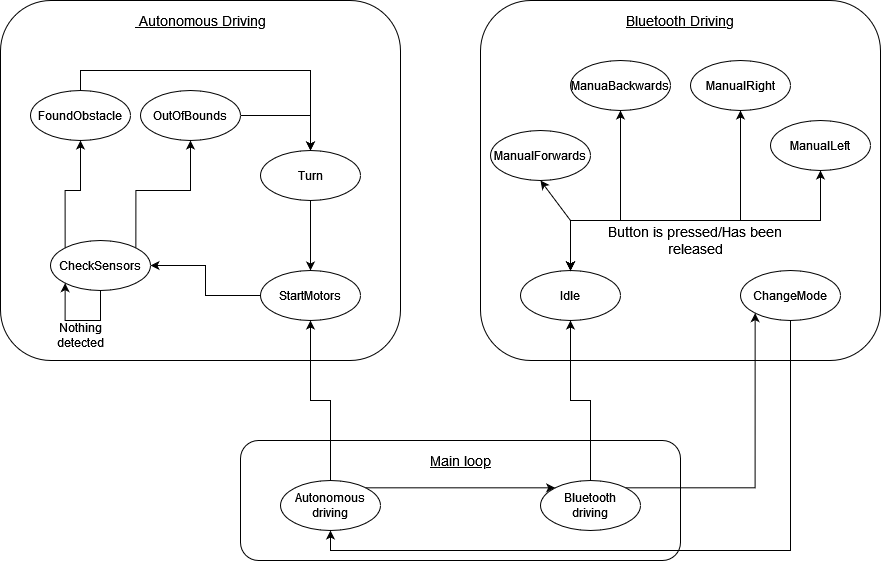
The mower uses both its Arduino Mega2560 and the Raspberry Pi zero w in tandem to fulfil the requirements set by the project. They are connected through serial UART communication with the use of a USB-B to USB-A cable which in turn is connected to a USB-B to microUSB converter. The Arduino board contains two state-machines, one for autonomous driving and one for manual driving with the mobile application. These state-machines are accessed through the main loop in the Arduino board.

After the initial setup the Arduino will remain on standby until it successfully has gotten a message from the Raspberry Pi. The Raspberry Pi begins it’s setup as well, connects to Wi-Fi and awaits a Bluetooth connection with the mobile application. When this connection between the application and Raspberry Pi is successful the main loop goes into the Bluetooth driving state-machine.

The Bluetooth driving state-machine’s neutral state is “idle” in which it remains stationary. When a direction button in the application is pressed the state will change to the corresponding state in the Arduino. When the button is released the state reverts back to “idle”. When the button to change mode is pressed the Arduino and Raspberry Pi goes into the automatic driving state-machine.

The automatic driving state-machine begins by starting the motors and then going into a state that checks the mower’s sensors. The mower remains in this state until either the line sensor detects the area limit or the ultra sensor detects an obstacle. When detecting the area limit the mower will go into the turning state. If an obstacle is found the Raspberry Pi will take a picture and then send it to Backend via Wi-Fi and then the mower will go into the turn state. The mower will turn for a set random time and then change states to start the motors and then once again start checking the sensors. If the application presses the button to change mode in any state, the mower goes into the Bluetooth driving state-machine.

The positioning is done via the Raspberry Pi. When the mower is in a driving state the Raspberry Pi updates the position forward. When the mower is turning it also sends at which new angle it’s turning with the gyroscope module to the Raspberry Pi. It then calculates what the new forward position is. The position is regularly sent to Backend via Wi-Fi.

The Arduino Mega2560 state-machines.

Detailed system design:

**MBot (C++)**

**Variables:**

* moveDirection –- An Enum for the “move()” function.
* mowerState\_t –- An Enum for the Bluetooth driving state-machine.
* distanceToObstacle –- An Int variable that represents when the ultra sensor is activated. Distance is in cm.
* autoState –- An Int variable that checks what state the mower is in the autonomous driving state-machine.
* bluetoothState –- A Char variable that checks what state the mower is in the Bluetooth driving state-machine.
* mode –- An Int variable that decides in the main loop whether the mower is in the autonomous driving mode or the Bluetooth driving mode.
* turnFlag –- An Int flag keeping track of if the mower is turning in the Bluetooth driving state-machine.
* autoTurnDirection –- A “moveDirection” variable keeping track what direction the mower should turn in the “autoTurn()” function.

**Functions:**

void \_delay(float milliSeconds):

Input: milliSeconds – The amount of time the delay shall happen. Is in milliseconds.

Output: None

Description: The program is paused for the time (in milliseconds) specified as parameter.

void move(moveDirection direction, int speed):

Input: direction – What direction the mower shall move. Uses the moveDirection enum as input.

speed – How fast the mower should move.

Output: None

Description: move function adjusts the motors depending on the direction.

void moveForward():

Input: None

Output: None

Description: When this function is called, the mower will drive forward

void moveBackwards():

Input: None

Output: None

Description: Call moveBackwards() to give the Mower more room to rotate.

void turnLeft():

Input: None

Output: None

Description: When this function is called, the mower will turn left.

void turnRight():

Input: None

Output: None

Description: When this function is called, the mower will turn right.

void stopMotors():

Input: None

Output: None

Description: Call stopMotors() to prevent the Mower from crossing the line.

void autoTurn():

Input: None

Output: None

Description: determines how long the robot will swing and in which direction depending on the Autoturn direction.

void isr\_process\_motorLeft(void):

Input: None

Output: None

Description:

void isr\_process\_motorRight(void):

Input: None

Output: None

Description:

int checkSensors():

Input: None

Output: Returns an Int that decides what state the autonomousDriving() will go into.

Description: This function will check the state of the sensor.

String getOrientation():

Input: None

Output: Returns the angle as a string.

Description: It provides the angle from the gyroscope module.

int autonomousDriving(int currentState):

Input: currentState – What state the function will go into.

Output: Returns an Int that respresents what the next state in autonomousDriving() will be.

Description: When this function is called, the Mower will run on its own. The Mower will check the state of linesensorStateGlobal and the ultrasonic sensor in this function.

void bluetoothDriving(char nextState):

Input: nextState – What state the function will go into.

Output: None

Description: When this function is called, the mower will be controlled by the application. Will check the data given from Bluetooth and let the Mower be in different states by changing the state of mower StateGlobal to forward, backwards, left, right or stop(idle).

void setup():

Input: None

Output: None

Description: Technically, void setup is a function that you define at the top of each program. The code that you want to run once when the program starts is enclosed in curly brackets. In this section, you can configure things like pinMode.

void \_loop():

Input: None

Output: None

Description: The loop() function will run over-and-over-and-over until the Arduino is reset.

**Raspberry pi (Python)**

**Variables:**

* running – A Boolean flag keeping track of if the program should continue running.
* mode – A String varying between “Manual” and “Automated” which controls which mode the mower is currently in and what commands it should be executing.
* reversing –- A Boolean flag keeping track of if the mower is reversing while in manual mode.
* turning – A Boolean flag keeping track of if the mower is turning while in manual mode.

**Functions:**

sendPositionRequest(x, y, sessionID, state, collisionFlag):

Input: x – The x-coordinate for the Mowers current position as an integer.

y – The y-coordinate for the Mowers current position as an integer.

sessionID – The ID of the current session as an integer.

state - defines if the session just started (“START”), is running(“RUNNING”), or has ended(“STOP”).

collisionFlag – A flag that should be set to True if a collision has occurred.

Output: None

Description: Function used to send a POST request regarding the mowers current position to the backend.

sendImageRequest(x,y):

Input: x – The x-coordinate for the Mowers current position as an integer.

y – The y-coordinate for the Mowers current position as an integer.

Output: None

Description: Function used to send a POST request with the most recently taken picture to the backend.

bluetoothInit():

Input: None

Output: client\_sock – The client socket connected to the rpi.

Description: Function used to initiate the Bluetooth server socket and awaits a connection from a client.

class CalculatePosition:

\_\_init\_\_(self):

Input: None

Output: None

Description: Initiates a CalculatePosition object. This class is used to be able to easily run and pause a separate thread that calculates the mowers current position through dead reckoning and trigonometry.

terminate(self):

Input: None

Output: None

Description: Function called to pause positional calculation in the separate thread. Simply sets a flag until next call to run.

run(self, speed, newDirection):

Input: speed – Defines the speed the mower is moving at.

newDirection – Defines the current direction the mower is moving in, should be an integer within the range of 0-360.

Output: None

Description: Function called to begin or continue calculating the mowers position in the separate thread.

class ReceiveBluetooth:

\_\_init\_\_(self, client):

Input: client – the socket client connected to the rpi.

Output: None

Description – Initiates a ReceiveBluetooth object. This class is used to be able to receive messages from the client socket from a separate thread running the background.

terminate(self):

Input: None

Output: None

Description: Function called to pause receiving Bluetooth messages in the separate thread. Simply sets a flag until next call to run.

run(self):

Input: None

Output: None

Description: Function called to begin receiving Bluetooth messages. When a message is received, it is stored in a variable and a flag is set to tell the main-thread that there is a new command. When the command has been handled by the main thread, the class can continue looking for new messages.