# turret

## Smarte systemer

**Nov 20, 2023**

# CONTENTS:

CHAPTER

# ONE

# TURRET

## 1.1 Course

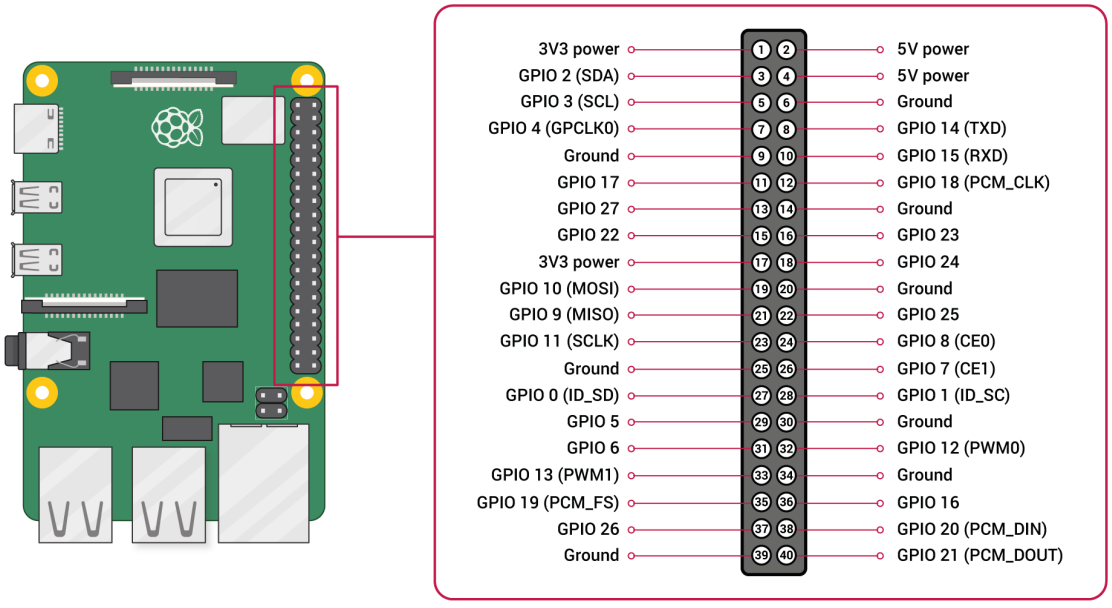
## 1.2 Weekly blog

## 1.3 Jupyter training notebook

## 1.4 Docstring

Docstring example

## 1.5 Raspberry pi pinout



### 1.5.1 picontroller

**class** turret.picontroller.**PiController**(*shared_coord: SharedVar*, *gui:* GUI, *camera:* Camera)

Bases: object

PiController class. Takes shared variable, a GUI object and a Camera object in constructor. Has variables for the all threads and delegates between them. Works as the brain between: 1. detection model and motor 2. GUI and motor

Calculates the necessary steps and communicates to the connected mcu.

**start**()

Function to call from main to start the application. Starts threads to camera, model and piControllers own run function.

**Parameters**
**None** –

**Returns**
None

**pixel_to_step_pitch**(*pixels: int*)

Calculates the number of steps needed for motors to move to detected object in y

**Parameters**
**pixels** (*int*) – Number of pixels, y, between camera center and object

**Returns**
Number of steps to pitch motor

**Return type**
_type_

**pixel_to_step_azimuth**(*pixels: int*, *fov: int*, *microstep: int*)

Calculates the number of steps needed for motors to move to detected object in x

**Parameters**
**microstep** (*int*) – Which microstep setting for azimuth motors

**Returns**
Number of steps needed to azimuth motors

**Return type**
_type_

**move_turret**(*leftMove: bool*, *rightMove: bool*, *upMove: bool*, *downMove: bool*)

Function that moves turret if Autoaim is toggled off. Reads arrow button values from GUI

**Parameters**

- **leftMove** (*bool*) – Button push from GUI
- **rightMove** (*bool*) – Button push from GUI
- **upMove** (*bool*) – Button push from GUI
- **downMove** (*bool*) – Button push from GUI

**move_to_target**(*tolerance*)

**The function that autonomously converts detectoion coordinates**
and sends the necessary steps to the azimuth and pitch motors for movement. 1. Gets necessary coordinates from shared variable 2. Calls the necessary functions for calculating steps 3. Communicates with mcu connected to motors

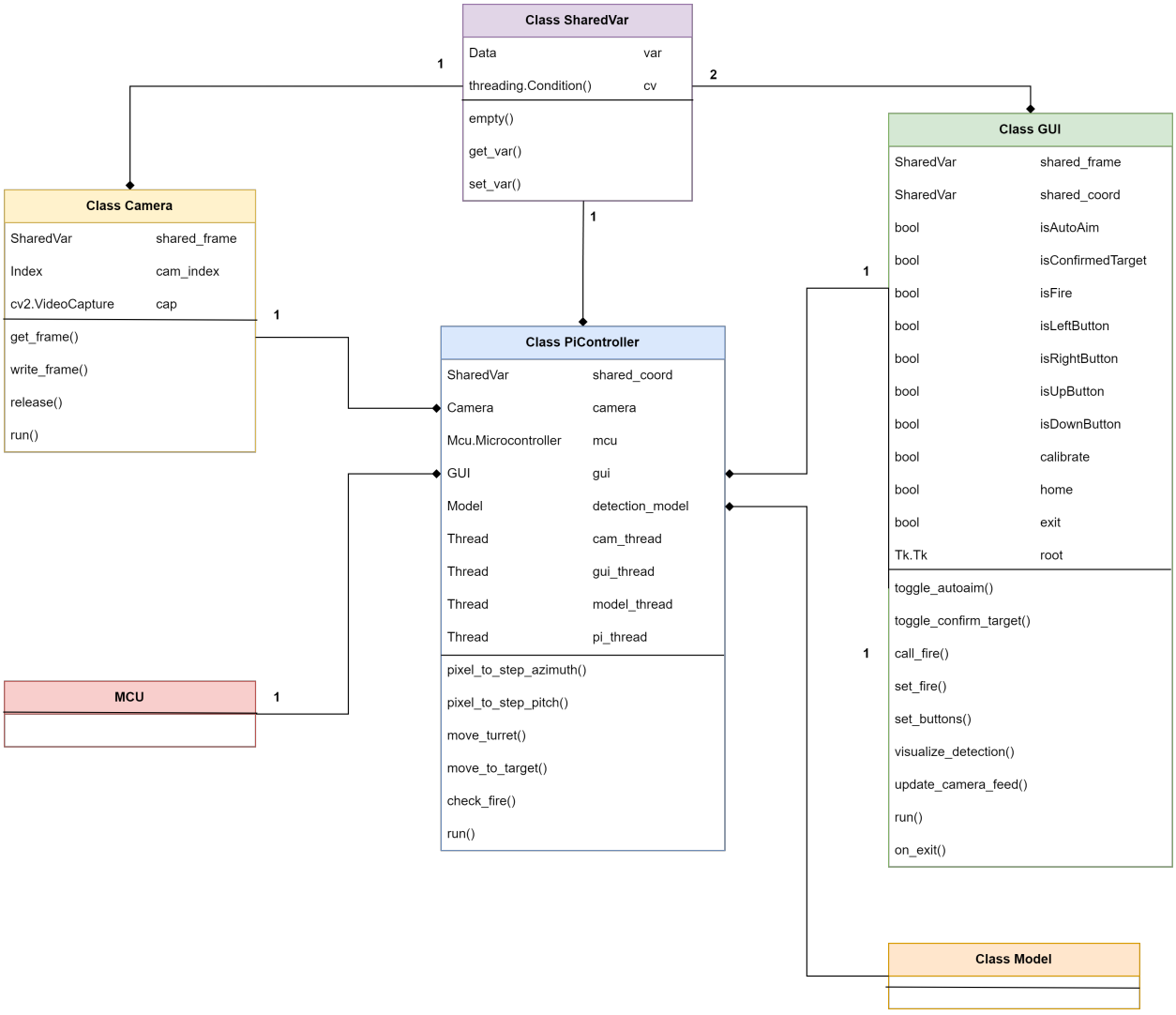
**Parameters**
**tolerance** (*_type_*) – The tolerance for stopping the movement in pixels

**calculate_azimuth_steps()**

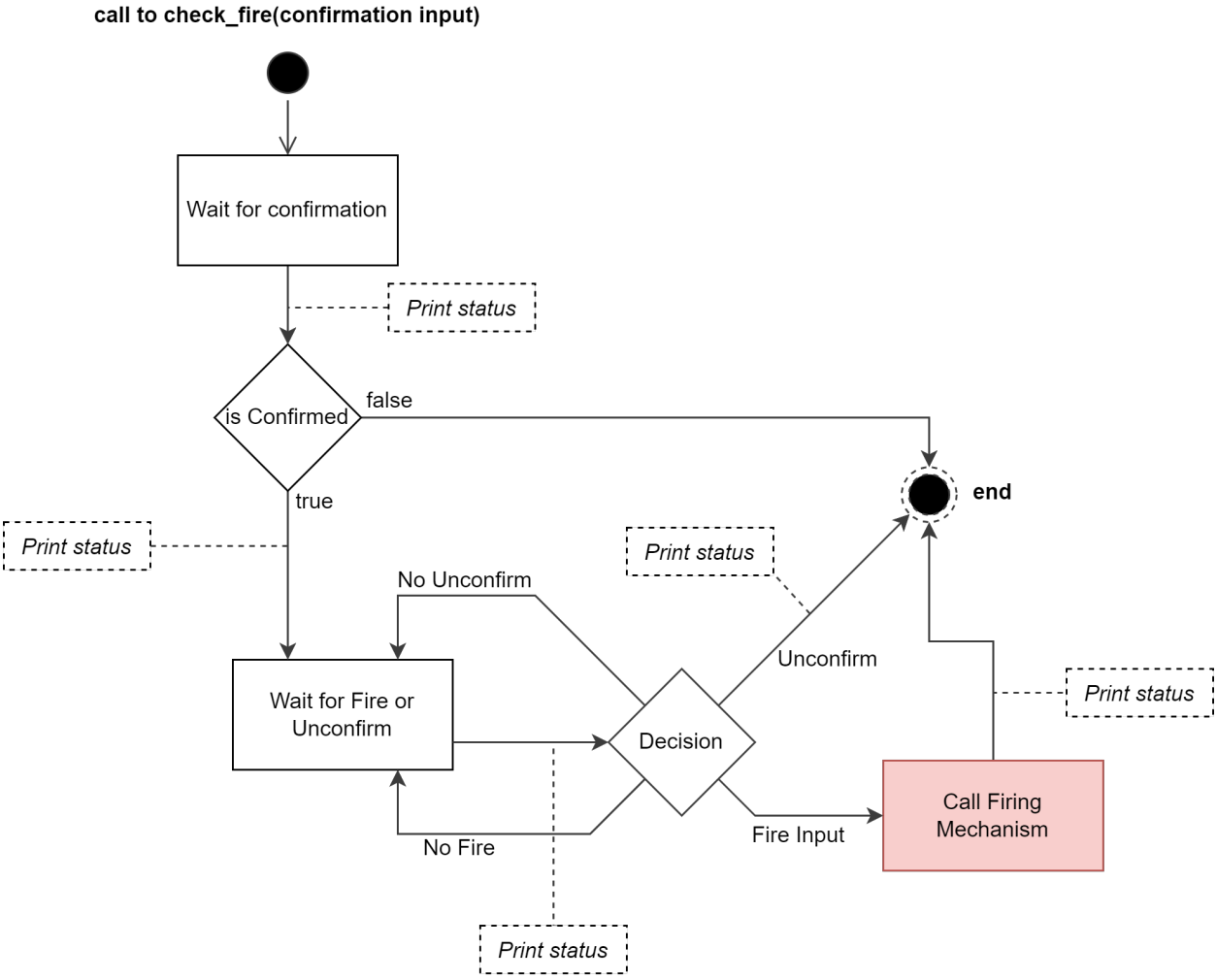
**check_fire()**
Fire procedure. Checks if target is confirmed in GUI and then if the correct order of operations communicates with the trigger motor.

**run()**
The piController thread-function. A while-loop that runs constantly if the application is not exited.

#### 1.5.1.1 Class Diagram



#### 1.5.1.2 Fire-logic




### 1.5.2 Graphical user interface

**class** turret.gui.**GUI**(*sharedFrame: SharedVar*, *sharedCoor: SharedVar*)

Bases: object

The GUI class. Takes shared variables for the frame and the coordinate. Creates buttons for movement, autoaim-toggle, confirm target, fire, calibrate and home in constructor. Has member variables for button pushes.

**toggle_autoaim**()

Function thats called when toogle Autoaim is pushed. Changes member variable isAutoaim between true and false when button is pressed.

**toggle_confirm_target**()

Function thats called when Confirm Target is pushed. Changes member variable isConfirmedTarget between true and false when button is pressed.

**call_fire**()

Function thats called when Fire button is pushed. Sets Fire to true when button is pressed.

**set_fire**()

**set_left_button**()

Function thats sets isLeftButton to true when pushed as long as isAutoaim is toggled off

**set_calibration**()

Function that sets calibration to true when pushed

**activate_home**()

Function thats sets home to true when pushed

**set_right_button**()

Function thats sets isRightButton to true when pushed as long as isAutoaim is toggles off

**set_up_button**()

Function thats sets isUpButton to true when pushed as long as isAutoaim is toggles off

**set_down_button**()

Function thats sets isDownButton to true when pushed as long as isAutoaim is toggles off

**visulize_detection**(*coordinate: Detection*, *frame: Mat | ndarray[Any, dtype[generic]]*) → Mat | ndarray[Any, dtype[generic]]

Function that connects the coordinates for the object and the frame from the camera and displays it

**Parameters**

- **coordinate** (*Detection*) – The coordinate for the detected object
- **frame** (*cv2.typing.MatLike*) – The frame from the camera

**Returns**
Returns the frame with the connected rectangle around the detected object

**Return type**
cv2.typing.MatLike

**update_camera_feed**()

The function thats updates the camera feed and displays it in the GUI. Function thats also resizes the camera feed.

**on_exit**()

Exit function. Destroys root-thread when called.

**run**()

Run-function. Updates the camera-feed for every frame and runs the loop for the root-thread

### 1.5.3 PID

**class** turret.PIDRegulator.**PIDRegulator**(*Kp*, *Ki*, *Kd*)

Bases: object

**integral**

Class for PID-regulator for smooth motor-movement.

**Parameters**

- **Kp** – Proportional constant - Needs to be tested for system
- **Ki** – Integral constant - Needs to be tested for system

- **Kd** – Derivative contants - Needs to be tested for system

**calculate**(*setpoint*, *current_pos*)

Calculates position based on wanted position and current position

**Parameters**

- **setpoint** – Wanted value for position
- **current_val** – Current value for position

### 1.5.4 Camera

**class** turret.camera.**Camera**(*sharedFrame: SharedVar*, *cam_index=0*)

Bases: object

The Camera class. Takes the shared variable for frames in constructor and an index for camera (default = 0) Opens the camera feed on camera in constructor.

**get_frame**()

Reads the camera frames from connected camera

**Returns**
The frame captures

**Return type**
_type_

**get_resolution**()

Get function for camera resolution

**Returns**
Returns a tupple of width and height in pixels

**Return type**
_type_

**write_frame**()

Function that writes frame to the shared variable

**release**()

Release function for connected camera

**run**()

The run function for the camera thread

### 1.5.5 Detector

Detector models were created using this Jupyter notebook.

**class** turret.detector.**Detector**(*frame: SharedVar*, *coordinates: SharedVar*, *model: str = '../object_detection/models/turret-syndrome-efficientdet_lite1.tflite'*, *threads: int = 2*, *max_results: int = 10*, *score_threshold: float = 0.7*)

Bases: object

**set_coordinates**(*image: Mat | ndarray[Any, dtype[generic]]*)

**Sets coordinates for detected objects in SharedVar object**
Format: ((x1,y1,x2,y2), class_name)

**Parameters**
**image** – frame from stream to process

**run**()
Function to continuously run.

### 1.5.6 Communication

**class** turret.communication.**Communication**(*server_address: str*, *server_port: int = 65000*, *max_buffer_size: int = 65500*, *image_format: str = '.jpg'*)

Bases: object

**send_frame**(*frame: Mat | ndarray[Any, dtype[generic]]*) → None
Send individual frames

**Parameters**
**frame** – Frame from video stream

**Raises**
**Exception** – Unable to encode frame

### 1.5.7 Motor

Interface for TB6600 motor drive

"This is a professional two-phase stepper motor driver. It supports speed and direction control. You can set its micro step and output current with 6 DIP switch. There are 7 kinds of micro steps (1, 2 / A, 2 / B, 4, 8, 16, 32) and 8 kinds of current control (0.5A, 1A, 1.5A, 2A, 2.5A, 2.8A, 3.0A, 3.5A) in all. And all signal terminals adopt high-speed optocoupler isolation, enhancing its anti-high-frequency interference ability."

This driver works by pulsing a signal on and off(50% on, 50% off) with a frequency of up to 13 KHz. To run the motor drivers at a higher frequency, the motor driver needs higher voltages. So to run at 13 KHz, the driver needs up to 40V.

**Note:** Maximum pulse frequency of 13 KHz. To use higher frequencies and speeds, the driver needs higher voltages.

#### 1.5.7.1 Datasheet

DFROBOT

#### 1.5.7.2 DIP-switch configuration

### Microstep control

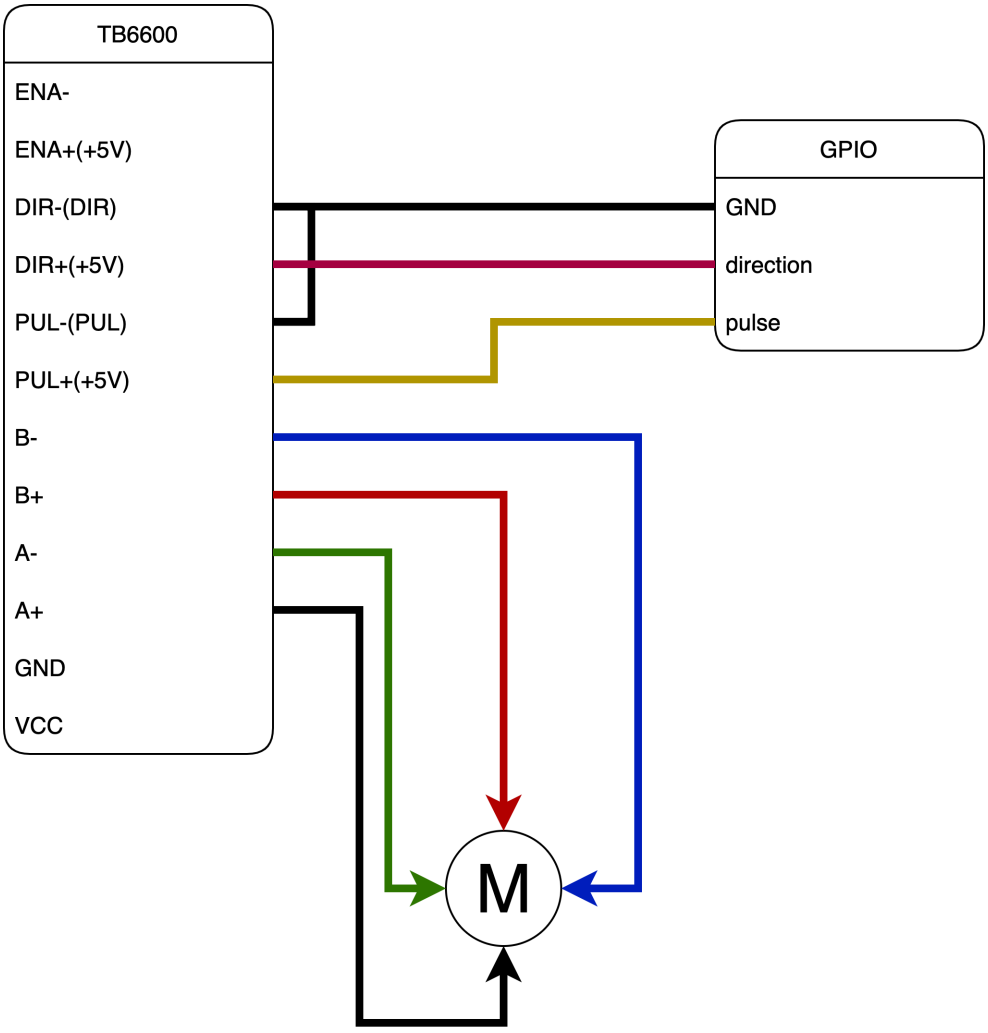
Microstep determines the step angle of the driver/motor configuration.

Stepper motors have a step angle which specifies what angle each step is. The microstep determines how much this step will be split up. The resulting step angle can be calculated in the following formula.

$$Angle\ per\ step = \frac{Motor\ angle}{Microstep}$$

### Current control

#### 1.5.7.3 Wiring



#### 1.5.7.4 Reference

class turret.motor.**Direction**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum for directions

**COUNTERCLOCKWISE = 1**

**CCW = 1**

**CLOCKWISE = 0**

**CW = 0**

class turret.motor.**Motor**(*pulse_pin: int*, *direction_pin: int*, *frequency: int*, *microstep: str = '32'*)

Bases: object

Constructor for motor driver

**Parameters**

- **pulse_pin** – Pin used to pulse
- **direction_pin** – Pin used to assign direction
- **frequency** – Frequency to pulse, higher -> faster.
- **pid_regulator** – PID used for calculating necessary movement
- **microstep** – Which mirostep setting. Defaults '200'

**get_microstep**() → int

**move_to_target**(*object_coordinates*, *cam_center*, *tolerance*)

**drive**(*steps: int*, *direction:* Direction *= Direction.CLOCKWISE*)

Drive motor in one direction for a number of steps

**Parameters**

- **steps** – Number of steps to rotate
- **direction** – Direction to rotate. Defaults to Direction.CLOCKWISE.

**drive_pwm**(*steps: int*, *direction:* Direction *= Direction.CLOCKWISE*)

**drive_revolution**(*revolutions: int*, *direction:* Direction *= Direction.CLOCKWISE*)

Perform full revolutions

**Parameters**

- **revolutions** – Number of revolutions to perform
- **direction** – Direction to rotate in. Defaults to Direction.CLOCKWISE.

**set_frequency**(*frequency: int*)

**get_period**() → int

**get__steps_per_revolutions**()

**__get_revolutions**(*microstep: str*)

Matches microstep setting to steps per revolution

**Parameters**
**microstep** – DIP switch setting

**Returns**
Number of steps per revolution

### 1.5.8 Interpolation

Implements two different interpolation techniques:

- Linear
- Polynomial

**Note:** Both are dependent on equally spaced x-values to work optimal.

#### 1.5.8.1 Linear

This interpolation works by finding the smallest interval for $y_a$ and $y_b$ so that $y$ is $y \in [y_a, y_b]$

The linear interpolation is then calculated as:

$$y = y_a + (y_b - y_a)\frac{x - x_a}{x_b - x_a}$$

#### 1.5.8.2 Polynomial

Polynimal interpolation uses Lagrange polynomial calculation. This method works by calculating basis polynomial for each value set. These basis polynomials are summed togheter to produce an estimated y-value.

$$P(x) = \sum_{i=1}^{n} y_j \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_i}{x_j - x_i}$$

Source

**class** turret.interpolation.**Interpolation**(*calibration_file: str*, *x_column: str*, *y_column: str*)

Bases: object

**linear_interpolation**(*x: float*) → float

Calculates linear interpolation for y given x

**Parameters**
**x** – x-value

**Returns**
Returns interpolated y-value

**polynomial_interpolation**(*x: float*) → float

Calculates interpolation polynomial

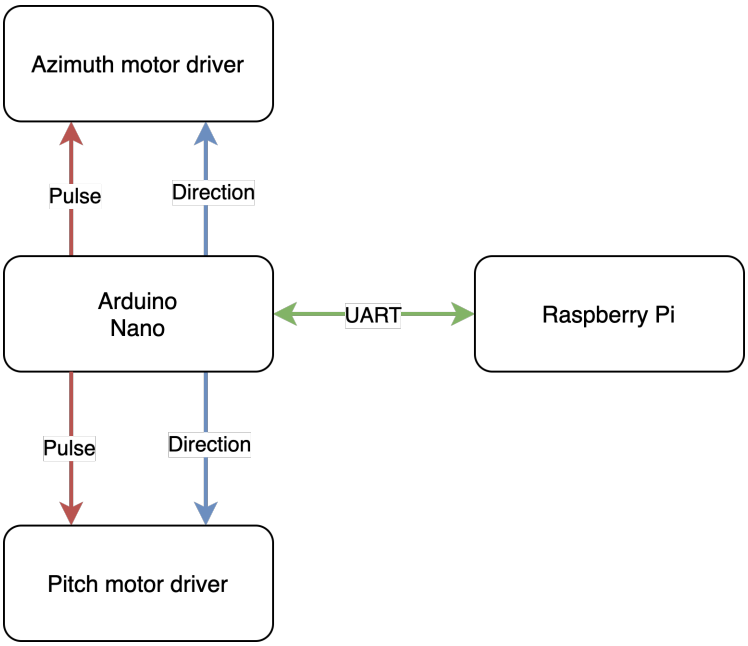
**Parameters**
**x** – value

**Returns**
Interpolated y-value

### 1.5.9 MCU

#### 1.5.9.1 Communication interface for Arduino nano

Driver for communicating and relaying commands to external microcontroller.




**class** turret.mcu.**Microcontroller**(*baud*, *signature*)

Bases: object

**check_for_response**()

Checks if microcontroller has responded.

**shoot**()

Function to send message that will make the mcu trigger shooting mechanism. Microcontroller should respond with "Fired" or "Cannot fire, in reload state" depending on its state.

**send_position**(*azimuth_steps: int*, *pitch_steps: int*)

Send position to microcontroller.

**Parameters**

- **azimuth_steps** – steps to write to azimuth motors
- **azimuth_direction** – direction for azimuth motors.
- **pitch_steps** – steps to write to pitch motor
- **pitch_direction** – Direction for the pitch motor to drive in.

**calibrate_pitch**()

Set new calibration point.

**home_pitch**()

Drive pitch to calibrated position

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## t

# INDEX

## R



## S



## T



## U



## V



## W