CAST Software Guide

Pending Copyright Information

Last Revised: 2022.05.1

Kenneth Grau

- Includes
  - Software Architecture
  - Sensor/Actuator Communication
  - Arm Obstacle Avoidance

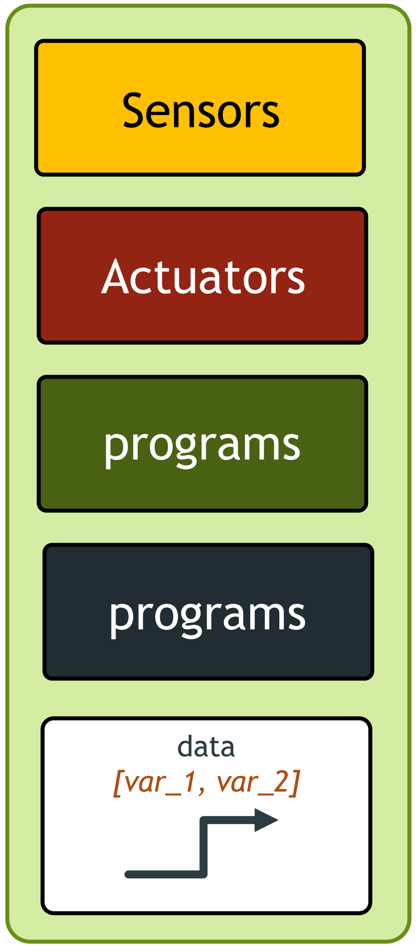
Software architecture

- Includes
  - Parts of each software file
  - How programs interact with each other
  - How programs interact with hardware
  - Sensor software vs. actuator software

- Language
  - This project is designed as a payload module for the SCUTTLE so it has been programmed similarly in Python3 on an embedded Linux platform on the supplied Raspberry Pi. The software has written in a similar format to the SCUTTLE to keep congruity with the two systems.

- It would be beneficial to review and be familiar with SCUTTLE documentation prior to reviewing this guide as the SCUTTLE robot is a major subsystem of the project.

Software architecture – Introduction



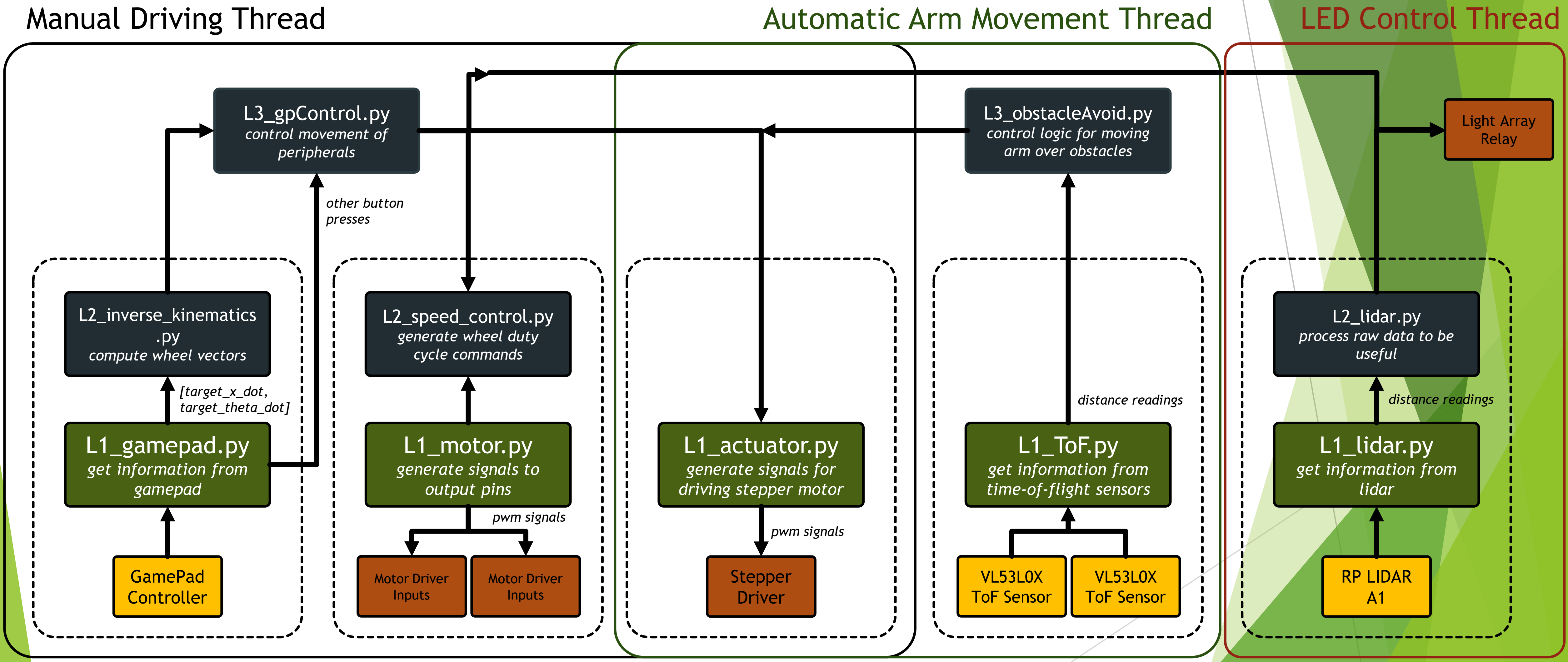
The **blocks in yellow** are sensors, and the items in red are actuators or other outputs. The level-2 **blocks in green** are specific to the hardware platform (beagle, pi, etc) and perform communication with the low level devices. **The blocks of level2 and above are non-hardware specific.**

Each block aside from sensors and actuators represent an individual python program. The purple text indicates what important information is passed between programs and the black arrows indicate (for the most part) what direction the data is flowing. Level 3 programs perform control logic and *(mostly)* receive data from the Level 2 programs.

This software structure is preferred in order to perform subsystem testing. The data flowing through the top level is minimal and can be replaced with artificial data in the even that a sensor is unavailable.

Most of this is pulled from SCUTTLE documentation with slight edits

Software Architecture - Overview



Manual Driving Thread
Automatic Arm Movement Thread
LED Control Thread
L3_gpControl.py
control movement of peripherals
L3_obstacleAvoid.py
control logic for moving arm over obstacles
Light Array Relay
other button presses
L2_inverse_kinematics .py
compute wheel vectors
L2_speed_control.py
generate wheel duty cycle commands
L2_lidar.py
process raw data to be useful
[target_x_dot, target_theta_dot]
distance readings
distance readings
L1_gamepad.py
get information from gamepad
L1_motor.py
generate signals to output pins
L1_actuator.py
generate signals for driving stepper motor
L1_ToF.py
get information from time-of-flight sensors
L1_lidar.py
get information from lidar
pwm signals
pwm signals
GamePad Controller
Motor Driver Inputs
Motor Driver Inputs
Stepper Driver
VL53L0X ToF Sensor
VL53L0X ToF Sensor
RP LIDAR A1

Libraries in Use

Python importing guidelines:

1. Each file should import files below it in the hierarchy, and not the files above it
2. Files may import non-CAST/SCUTTLE libraries as needed
3. If the Level 1 file has an imported external library, it doesn't need to be imported by the Level 2 file

Congruent with SCUTTLE importing guidelines

Libraries Utilized

Raspberry PI

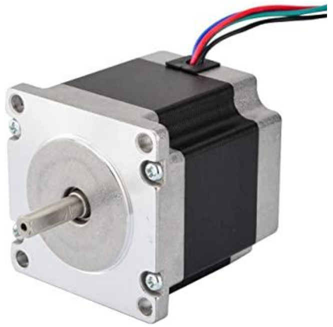
- gpiozero for controlling GPIO pins
- pigpio for controlling GPIO pins and providing access to hardware based PWM
- VL53L0X ToF Sensor integration library to provide distance reading functions
- adafruit_tca9548a for interfacing with multiplexor
- adafruit_rplidar for interfacing with lidar

Common Libraries

- os for making shell commands via python code
- time for keeping time
- threading for multithreading
- numpy for performing array math
- busio for communicating via I2C
- math for useful math functions

Actuator Information

- Actuator assembly consists of NEMA 23 Stepper Motor, STM-5045 Stepper Driver, Ball Screw and Rail Guides
  - Driver can handle signals up to 300kHz
  - Selected NEMA 23 Motor can handle speeds up to ~13 rps
  - Ball Screw has a linear travel per revolution of 5.08mm
  - There is no feedback in this system



- Precision Stepping – Generates square wave by manually stepping GPIO on/off

Equation to generate delay according to speed input

Where LDPR = linear distance traveled per revolution of ball screw

SPR = steps per revolution as determined by stepper driver

$$Delay = \frac{\frac{1}{\frac{desired\ speed}{LDPR * SPR}}}{2}$$

- PWM Generation – Uses hardware clock to generate PWM square wave

Equation to generate frequency for PWM according to speed

Where LDPR = linear distance traveled per revolution of ball screw

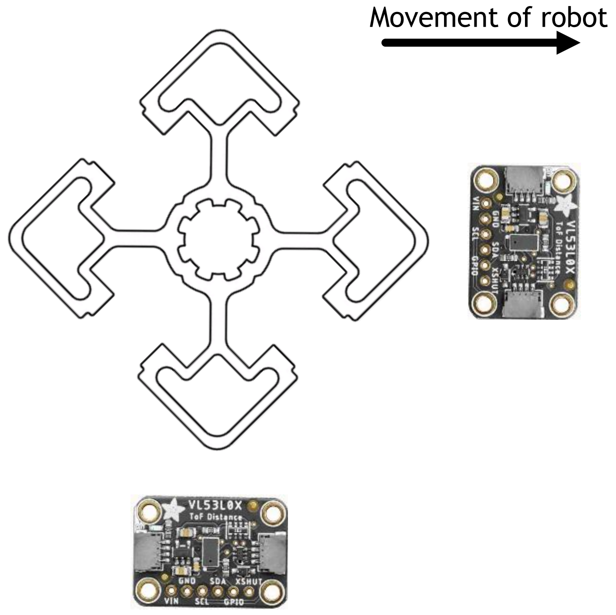
SPR = steps per revolution as determined by stepper driver

$$f = \frac{desired\ speed}{LDPR} \text{ x SPR}$$

PWM stepping uses the time.sleep() function to allow PWM to drive motor for certain time, and therefor certain distance. This function has a resolution of about 1ms. At max speed (9cm of linear travel per second), ~15 steps can occur giving a linear resolution of .09mm.

Concept of Operation - Arm

- Arm assembly consists of the arm housing LED arrays, 2 time of flight sensors, and actuator
  - Time of flight sensor oriented along the front plane and bottom plane of the arm
  - This was done to avoid having to use an array of sensors
  - This relies on the known length of the arm



Viewed as if you were looking down the arm

- Front Time of Flight senses obstacles in the path of the arm and executes simple check

If sensor returns a distance reading smaller than the length of the arm, actuate the arm up to avoid the obstacle. Continue to move the arm up until the sensor no longer returns a value less than the length of the arm

- Bottom Time of Flight senses objects under the arm and executes a simple check

If sensor returns a distance reading smaller than the length of the arm, stop the arm movement as there is something below the arm. In most cases, if there is nothing below the arm, the actuator continuously tries to return to its lowest point above the table.

Time of flight sensors work best with solid backgrounds and large opaque objects