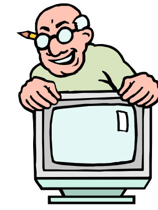
PixyBot2

Servo Calibration

CAUTION



Lithium batteries can be extremely dangerous, if not handled and cared for properly. This design does not include any form of current limiting circuit, like a fuse. So, care must be taken to ensure that the wiring guidelines are followed accurately, that checks are made for short-circuits, and that battery polarities are marked, and they are inserted the correct way round. Failure to do so, could result in an explosive fire.

Charging Practices: Always remove batteries from your project to charge them. Use a charger, designed for the battery used, and from a trusted supplier. Choose a flat, non-flammable surface to charge on, away from flammable materials. Never leave unattended when charging. Don't charge overnight. Monitor charging to ensure charge characteristics are as expected. Only pair batteries with similar characteristics. Do not overcharge, or leave charging for prolonged periods. This increases the risk of damage and fire.

Battery care & maintenance: Stop using a battery if it is swollen, damaged, dented or leaking. Never charge a damaged battery. Never allow a Lithium battery to discharge below 3.2 volts, as cell damage will occur. Avoid extreme temperatures. Do not charge or store batteries in very hot or cold environments. Don't cover batteries whilst charging, as this can trap heat, causing overheating.

In case of fire: Get out and stay out. If a fire starts, leave immediately, and call the fire brigade. For low voltage Lithium batteries, water is a safe extinguisher.

Built-in Monitoring: Most of my project designs include code, and circuitry, to monitor battery voltage, whilst in use. This code then seeks to alert the operator, when the battery has reached a critical low voltage, before shutting down power consuming circuitry; including the micro. Time should therefore be spent on calibrating this feature, as a precaution, for good battery management and maintenance.

Carefully dispose of batteries that damaged, or discharged below their critical voltage.

Why do we need to calibrate the servos?

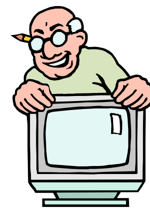
- No two servos are the same
- Servos can be damaged if not setup correctly
- Course calibration must be performed prior to the assembly process
- This sets approximate positions for the leaver arms
- Course calibration ensure servos are within mechanical limits
- Fine calibration determines min/max robot physical limits
- The ESP32 C++ code needs limit values in order to work accurately
- Hence, each robot has a unique set of calibrated values

Servo calibration is performed in two stages:

- Pre-set ensures mechanical parts are assembled correctly
- Fine calibration, performed during testing

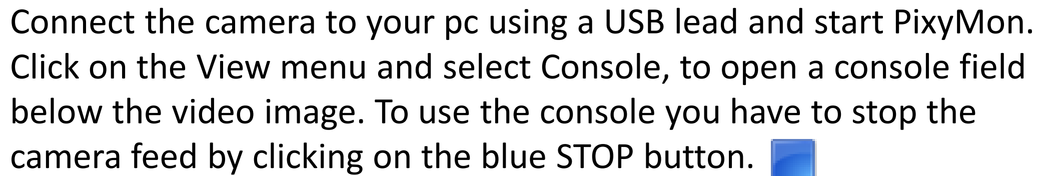
- Repeat this process for a given servo if it is ever replaced.

PixyMon



We start the calibration process by using the Windows PixyMon app to set the initial positions of the pan and tilt servos. This is done before the Pixy2 pan and tilt assembly is attached to the robot chassis.

Connect the camera to your pc using a USB lead and start PixyMon. Click on the View menu and select Console, to open a console field below the video image. To use the console you have to stop the camera feed by clicking on the blue STOP button.

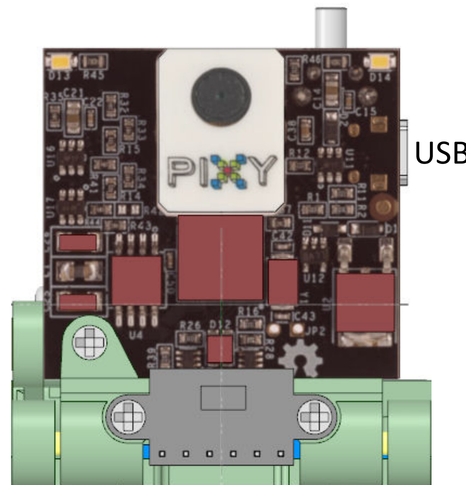


You can now enter commands to control the two servos and set their calibration positions for Pan and Tilt. Servo S0 rotates the camera (Pan) via the PWM0 signals, and servo S1 tilts the camera forwards via the PWM1 signals. Each servo can be sent a value ranging from 0 to 1000, which represents the PWM range of 1000 to 2000 µs, normally applied to servos. The command use is:

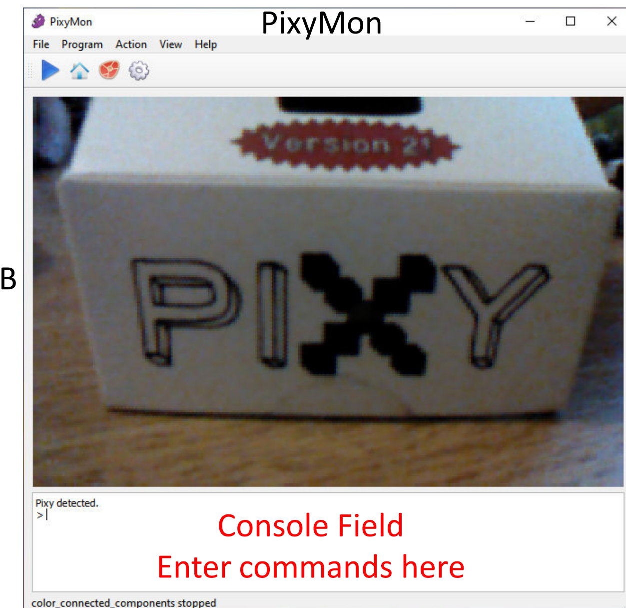
```
rcs_setPos  S  PWM
```

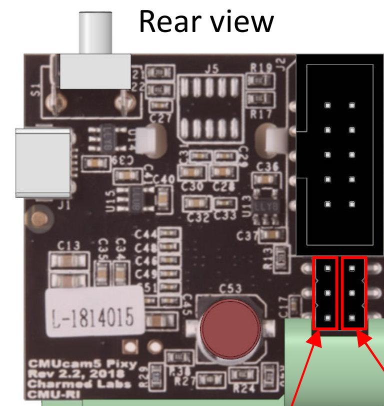
Where S is either 0 or 1 defining the servo, and the PWM is in the range 0 – 1000.

Set the pan servo S0 to its centre position by entering the following rcs_setPos 0 500 and press RETURN on the keyboard. Pixy2 should respond with response: 0 0(x0) meaning ok. Reposition the servo drive arm if not in its centre position (as close as possible). The positions of the splines on the servo drive shaft are unlikely to allow you to set the arm position as precisely as you would like.

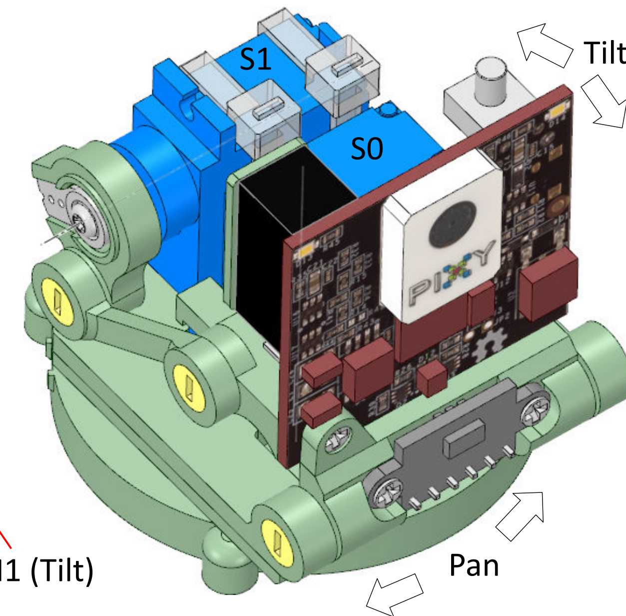


Front view

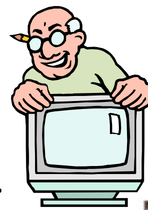




Rear view



PixyMon

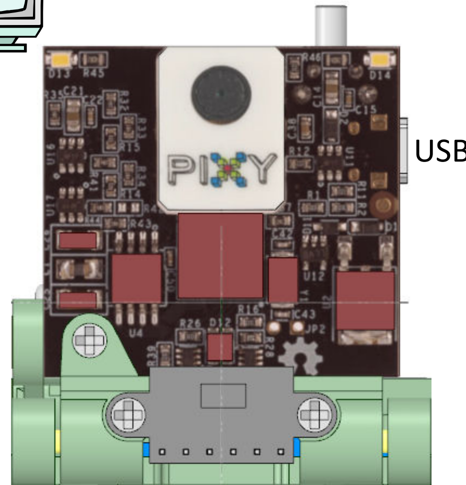


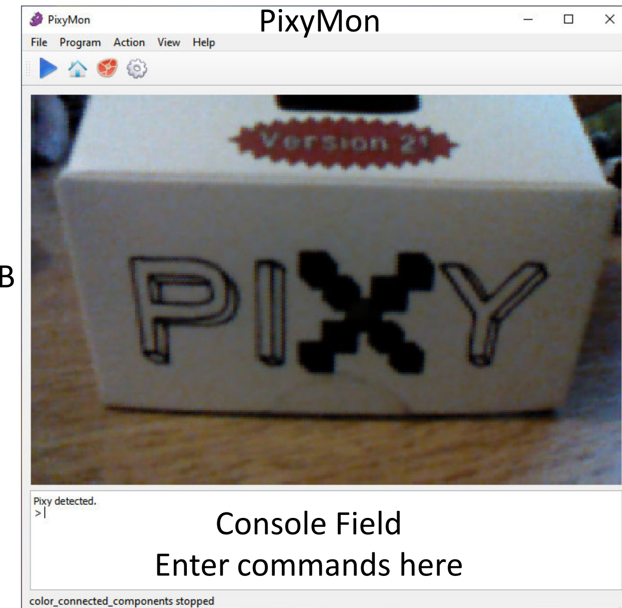
Now set the tilt servo S1 to its centre position by entering the following rcs_setPos 1 300 and press RETURN on the keyboard. Again, Pixy2 should respond with response: 0 0(x0) meaning ok.

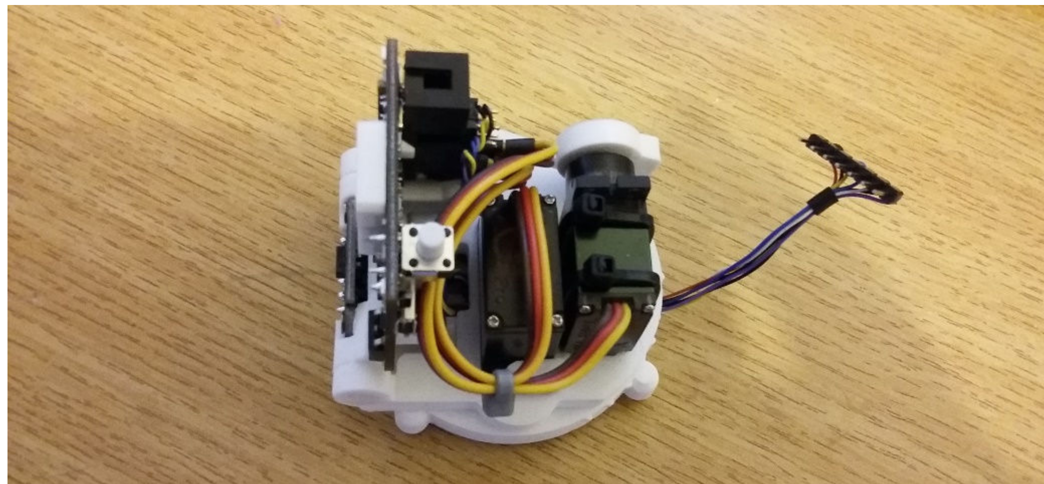
Reposition the servo S1 drive arm such that the Pixy2 camera is in the vertical position (as close as possible). Again, the positions of the splines on the servo drive shaft are unlikely to allow you to set the arm position precisely as you would like.

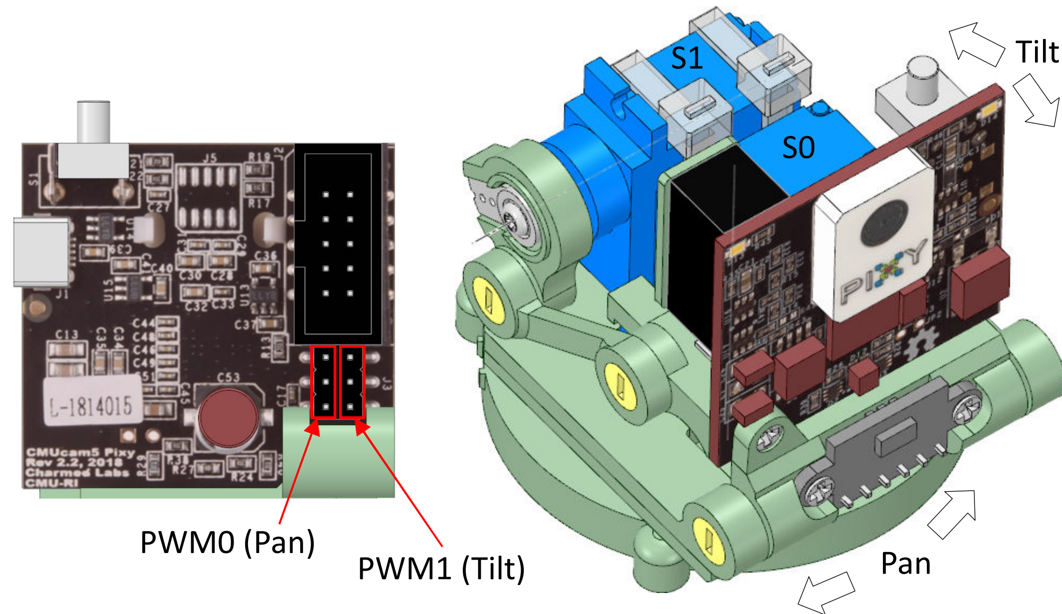
Note that the default power-on state PWM values for both servos is 500, so the camera should start near centre, but leaning forward. This is deliberate as, due to physical constraint behind the camera, the camera can tilt forwards further than it can backwards.

This completes the preliminary calibration and the camera assembly can now be wired and fitted to the robot chassis, as described in the wiring documentation.

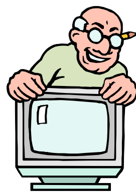




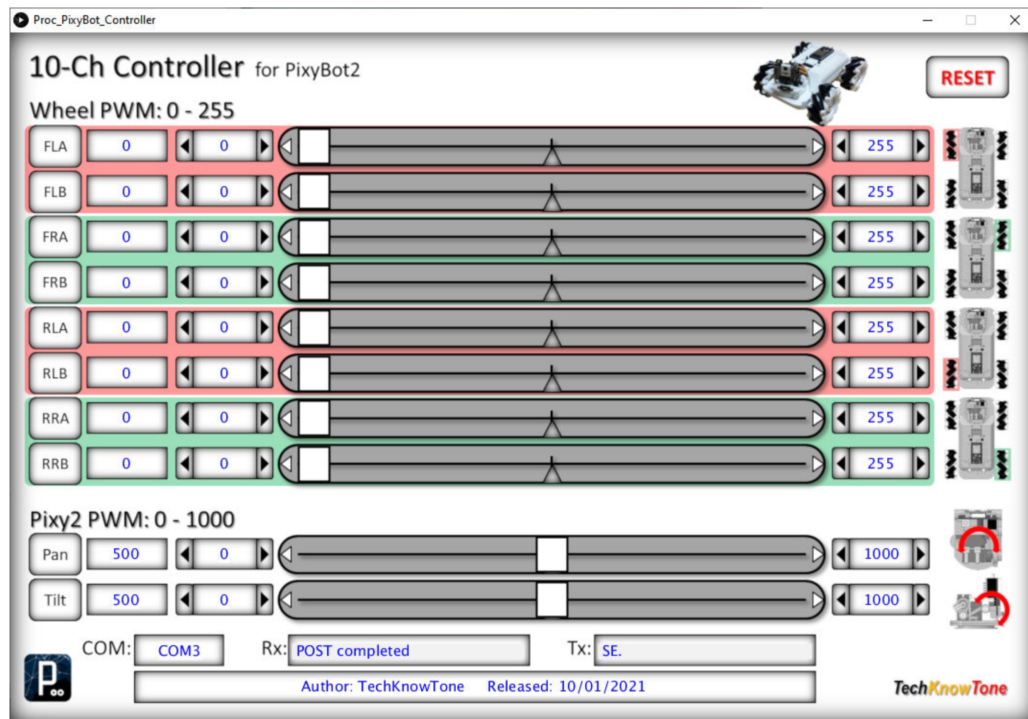




10-Channel Controller App



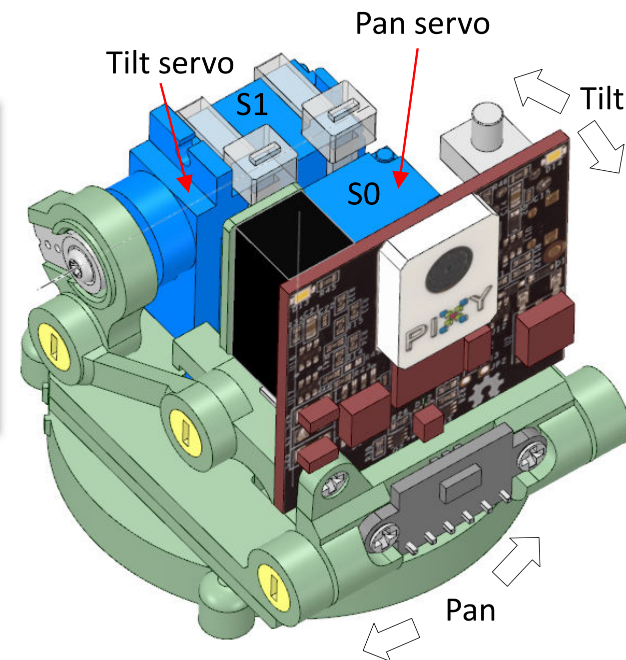
This custom Windows app enables you to calibrate the Pixy2 Pan and tilt servo motors, and gain a better understanding of the effect of the PWM signals being applied to the H-bridge motor drive systems.



The app connects to the ESP32 micro using the USB serial port, and transfers commands to it whenever sliders and buttons are adjusted in the app. The lower section of the app is for adjusting the Pixy2 pan and tilt servos. Clicking the buttons on the left toggles the serial commands ON/OFF. When ON, the slider button is read and the Tx commands will have a chevron '>' character in front of them, like: >ST8.SA524 which tells the micro to send the value 524 to the Pixy2 pan (S0) servo over the I2C interface. The header ST8 is used as it is the 9th slider on the app.

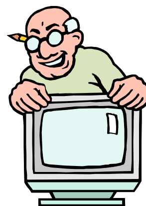
The pan and tilt slider are adjusted to determine values and limits that can be entered into the ESP32 micros code:

```
// define servo arrays
// [0] - Pixy2 Pan servo
// [1] - Pixy2 Tilt servo
//                    Pan,Tilt
int16_t servoCtr[] = { 554, 252}; // centre value sent to servos
int16_t servoInt[] = {   0,   0}; // servo tracking integrator
int16_t servoMax[] = { 947,1000}; // servo max pulses, Pan 60° left, Tilt leaning forward
int16_t servoMin[] = { 161,   0}; // servo min pulses, Pan 60° right, Tilt leaning backward
int16_t servoVal[] = { 554, 252}; // value sent to servos pan,tilt
```

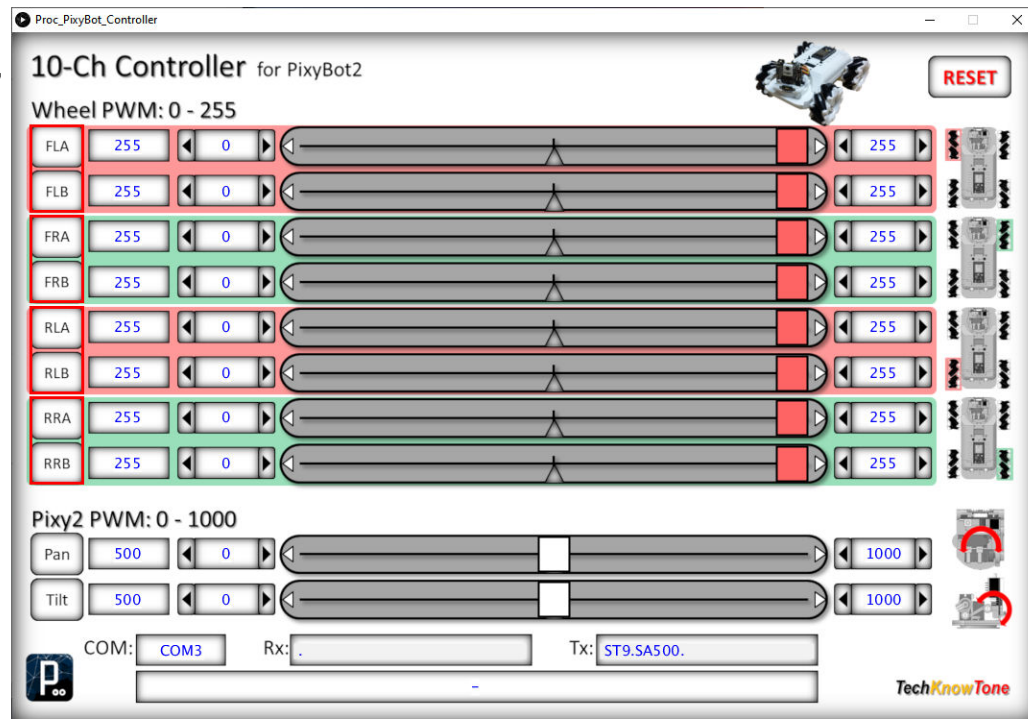


The above screen shot from the IDE, shows the values I determined and entered for each servo, which are preloaded and stored in arrays. Your values will be different to mine, and if you ever have a need to change the servos, then this process of calibration will need to be repeated.

Motor PWM Control



The apps sliders for motor control work slightly differently to that of Pixy2 servo control, because of the logic that needs to be applied to the H-bridge drivers. In Table 2 below, you will see that when PWM is being applied to one of the inputs, the other should be held in either a LOW (0) or HIGH (1) state, which corresponds to either a 0 or 255 PWM value. Hence, in the app if one slider is somewhere between 0 – 255, say 128, then the other slider for that pair can only be set to either 0 or 255.



You will find during experimentation that there appears to be much greater range of adjustment when one slider is at 255, than when one is held at 0, and that the speed is inversely proportional to the slider value. This is to do with how the H-bridge operates, and how the back emf of the motor is dissipated in the H-bridge circuit. Using two PWM signals from the micro we can drive the motor in either a clockwise or anticlockwise motion.

Table 1. H-Bridge Logic

xIN1	xIN2	xOUT1	xOUT2	FUNCTION
0	0	Z	Z	Coast/fast decay
0	1	L	H	Reverse
1	0	H	L	Forward
1	1	L	L	Brake/slow decay

In the C++ code we modify the drive signals to take account of this effect, and use a function which accepts values of 0 to +/-255, with the sign determining the direction of rotation. We pass four variable values into this function:

MotorPWM (FL, FR, RL, RR)

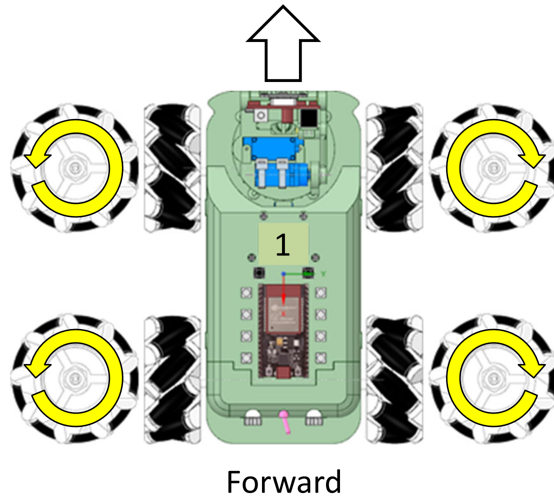
Table 2. PWM Control of Motor Speed

xIN1	xIN2	FUNCTION
PWM	0	Forward PWM, fast decay
1	PWM	Forward PWM, slow decay
0	PWM	Reverse PWM, fast decay
PWM	1	Reverse PWM, slow decay

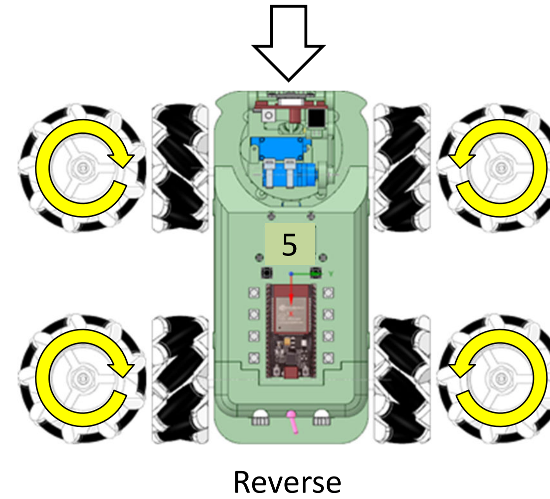
Where each variable will have a value between -255 and 255. If the value is 0 then the inputs to the H-Bridge will be set LOW to provide a braking effect.

Normal Drive n JoyMode

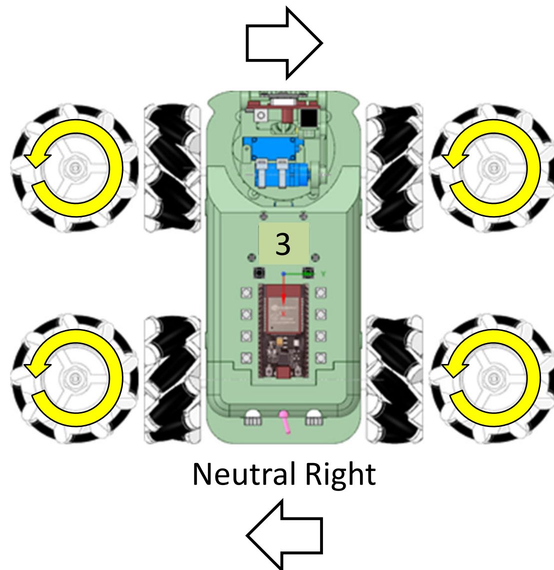
These diagrams show the direction of rotation for each wheel, needed to drive the PixyBot2 in a given direction.



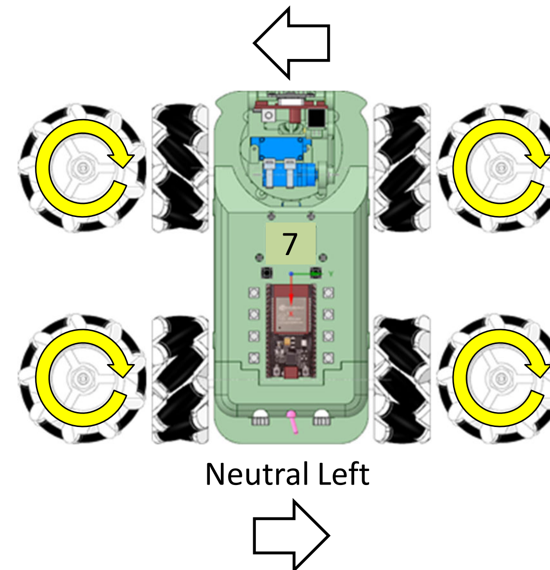
Forward



Reverse

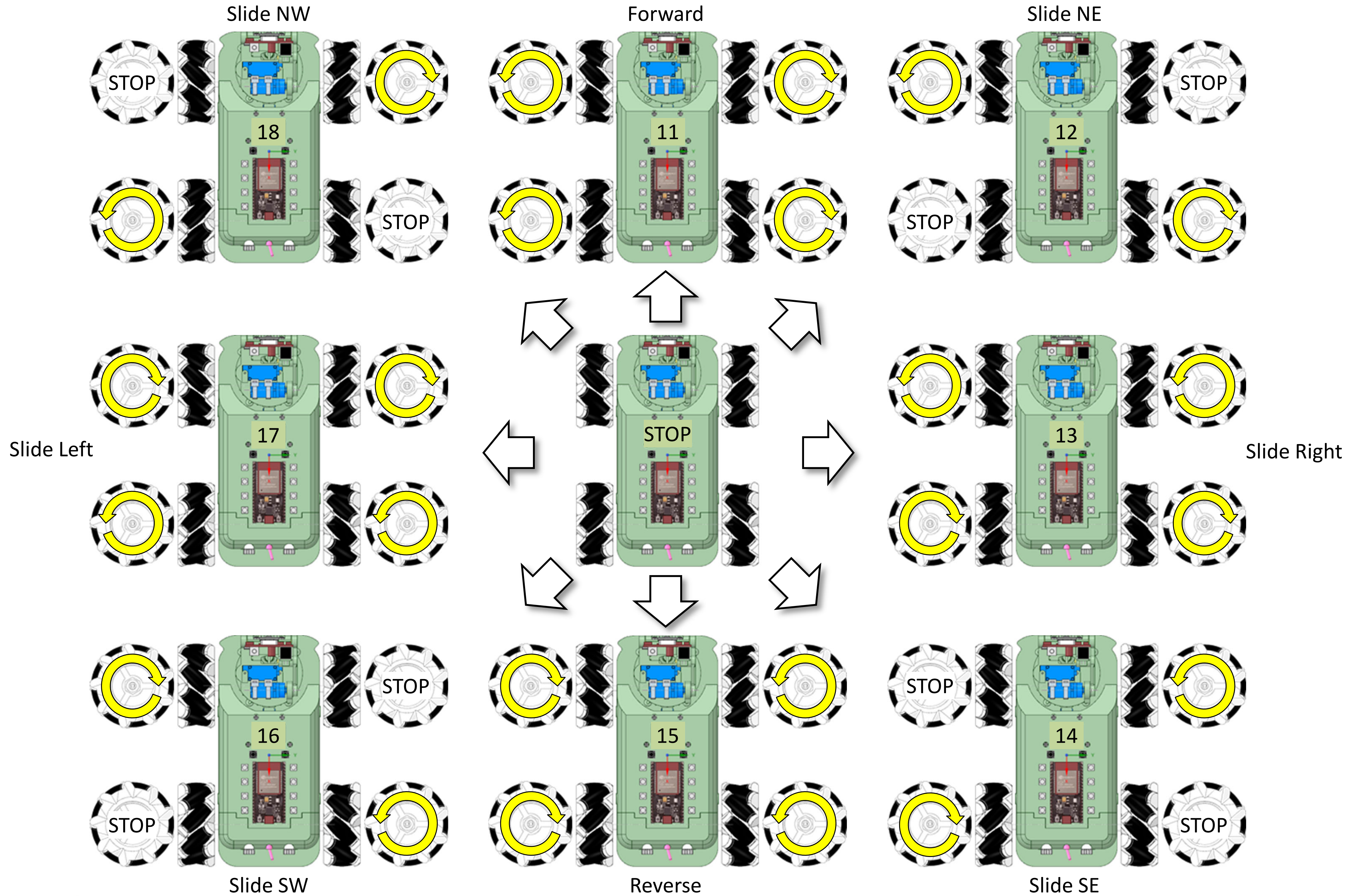


Neutral Right



Neutral Left

Mecanum Drive n JoyMode



Issue: 1.3 Released: 26/11/2025 TechKnowTone

Battery Voltage Health Monitoring

See 18650 discharge curve obtained from the internet. In this analysis both batteries are identical and connected in series,
Assume fully charged batteries max voltage is V_{BM} >= 8.2v max
I measured my fully recharge 18650 at 8.4v when connected and ON.
Set battery warning point at V_B = 7.00v
Set battery critical point at V_{BC} = 6.60v

ESP32 is powered from batteries connected to V_{in}.
3.3v at VADC == 4095 on 12-bit converter (4095 max).
If we use a 6k8 resistor feeding A0 and a 3k3 resistor to GND, we get a conversion factor of 10.1v == 4095 or 2.47mV/bit or 404.85
Using a Multimeter I determined the conversion factor needed to be reduced to 383.9 to display voltage correctly.

MAX: V_M = 8.2v, gives A0 = 3148 on ADC (V_M* 383.9)

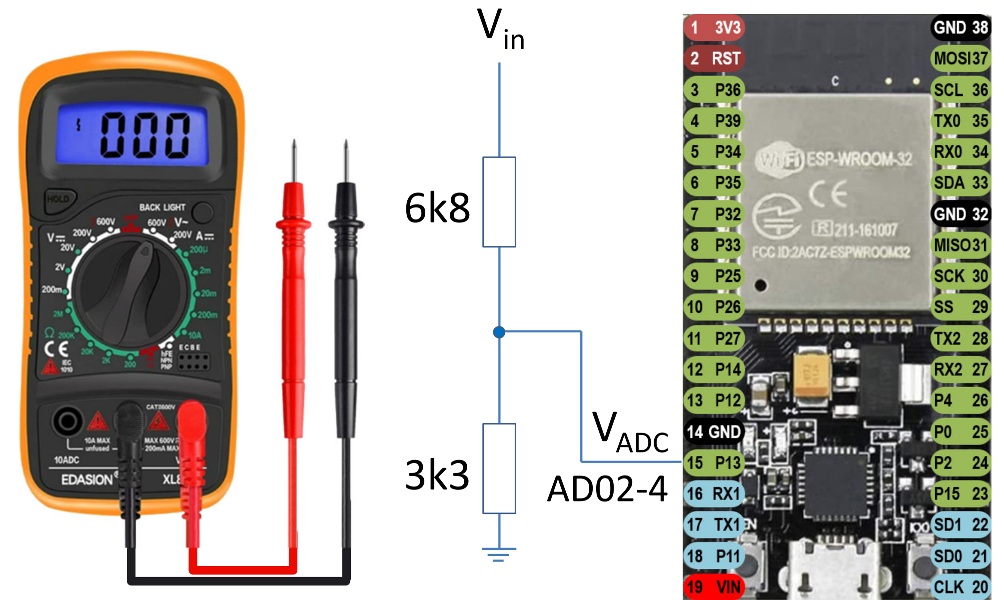
WARNING: V_B = 7.0v, gives A0 = 2687on ADC (V_B* 383.9)

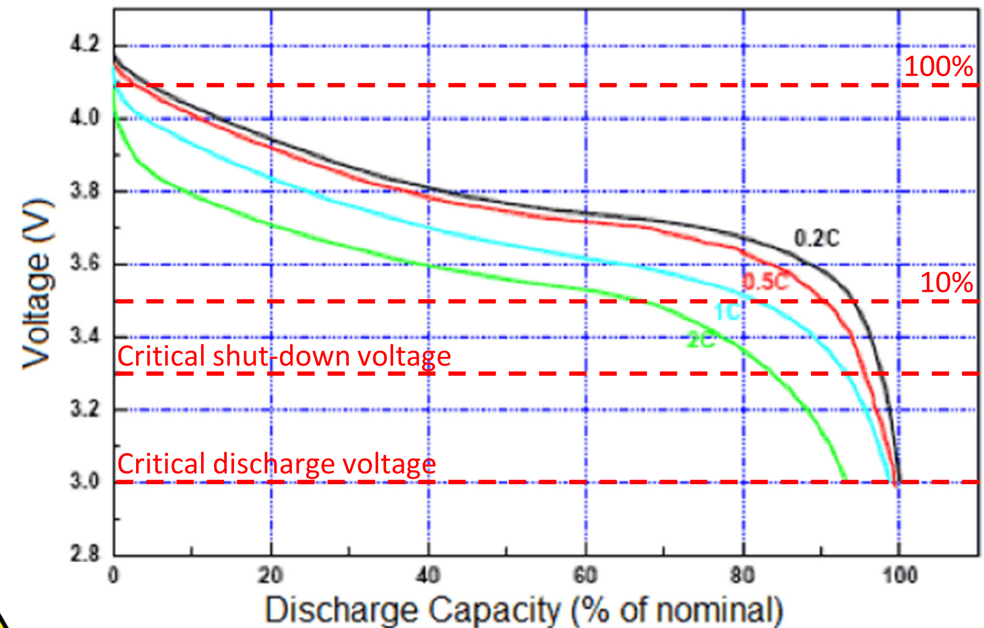
CRITICAL: V_{BC} = 6.6v, gives A0 = 2534 on ADC (V_{BC}* 383.9)

The code will sample the battery voltage on power-up to ensure it is sufficient, then at every 40ms interval, calculating an average (1/20) to remove noise.

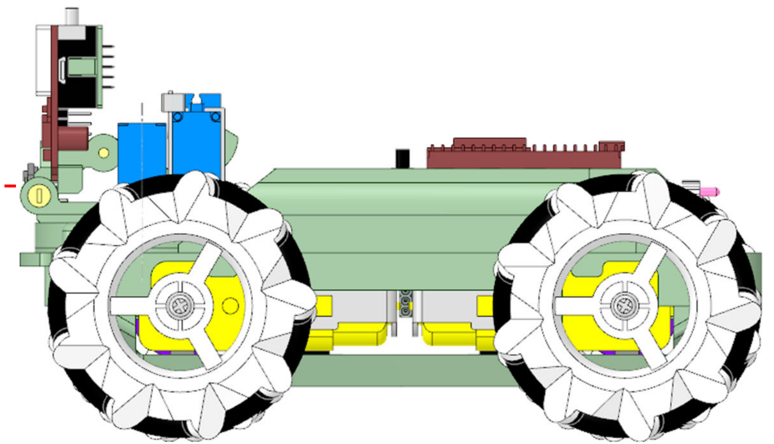
Given the relatively light current drawn I have assumed a linear discharge curve ranging from 8.2v (100%) to 6.6v (0%) capacity. The rate of discharge is monitored and used to actively predict the life of the battery in use.

Note: If connected to USB port with internal battery switched OFF the ADC will read a value 5 volts (A0 = 1919) or less. So if the micro starts with such a low reading it knows that it is on USB power.





VL53L1X Fixed & Rotating Sensor



30cm
2.5cm
Max.
335
Mid.
195
Min.
55

Speed

Back away
Reverse
Max.
Dead
Speed-up
Constant
Slow down
Dead
R3= 325
R2=180
R1=124
R0=55

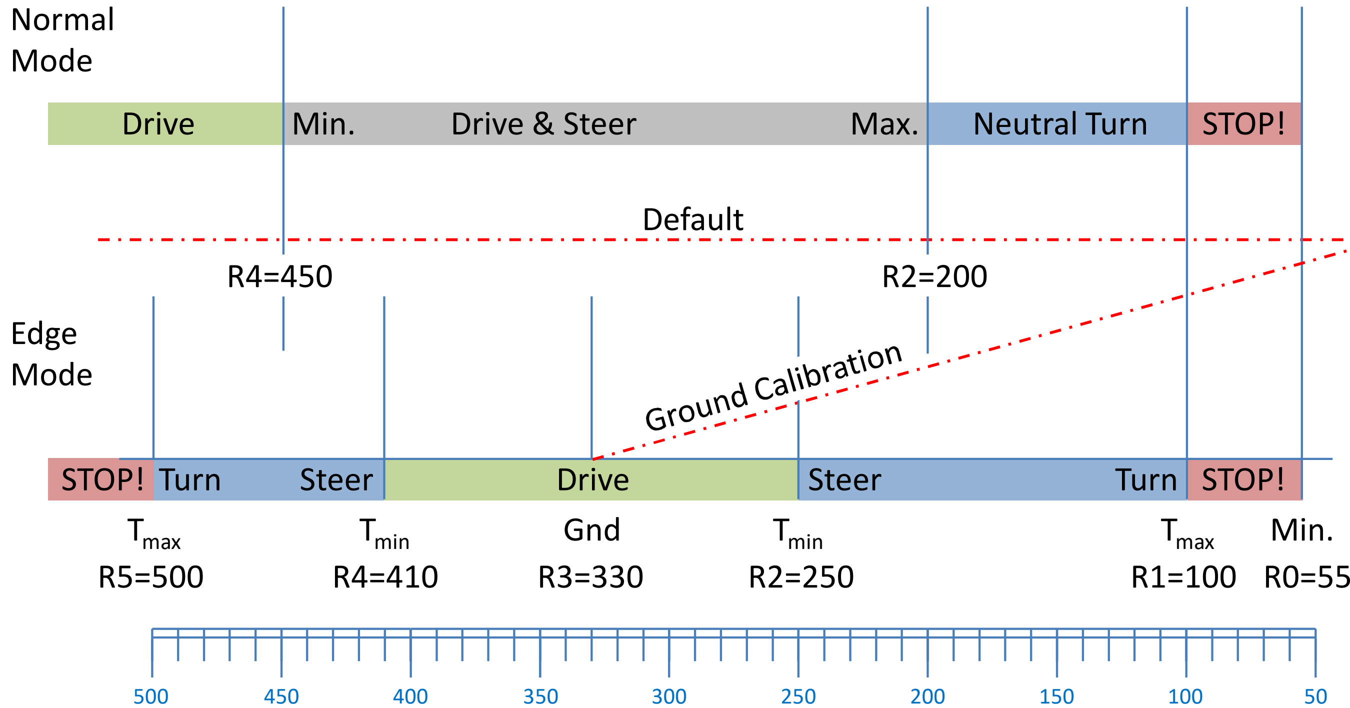
Target
Track
Forward
Reverse
Max.
Max.
Dead
Speed-up
Speed-up
Dead
Speed-up
Dead
R6=300
R5=250
R4=200
R3=170
R2=130
R1=90
R0=55
RM=335

300
250
200
150
100
50

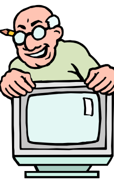
These are the values I determined from my VL53L1X sensor, and the speed maps I developed for the back away and target tracking functions.

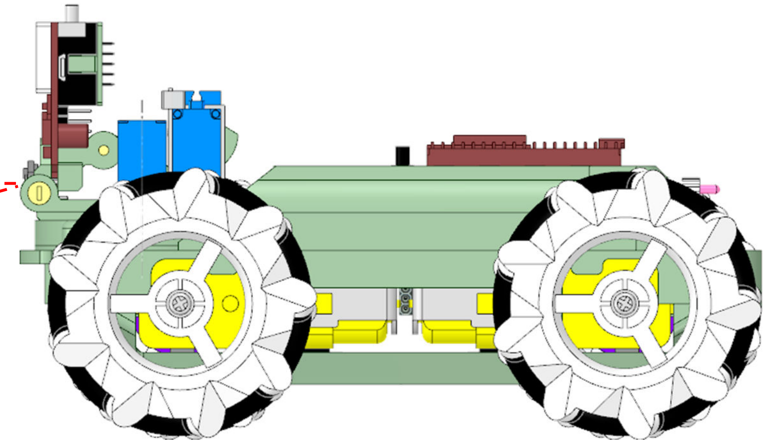
You should be able to find these values in the code and if necessary substitute the values you have determined from your sensor. Use the serial monitor in the IDE along with Serial.print() functions to display your values.

VL53L1X (LTOF) Autonomous Ranging



In normal autonomous mode, the LTOF sensor is looking straight ahead. If there is nothing within range it drives forward.





Tilting the sensor forwards enables us to detect the edges of a table, when running in autonomous edge mode.

These are the values I determined from my VL53L1X sensor, and the 'turn' maps I developed for the two autonomous modes, normal and desktop edge.

In edge mode, the calibration tilt 'Gnd' point is determined automatically by the code as we enter autonomous mode. Looking straight ahead, with nothing in the field of view, the ranging function should return a max. value of 500mm. If that is the case, then the LTOF sensor will be tilted forwards progressively, until the range drops to 350mm, which is said to be the ground point. It will remain at this tilt angle for the remainder of autonomous mode.

You should be able to find these values in the code, and if necessary substitute the values you have determined from your sensor. Use the LTOF data display mode and the OLED Mirror app to display profiles of what is in the field of view, in order to establish your own values.

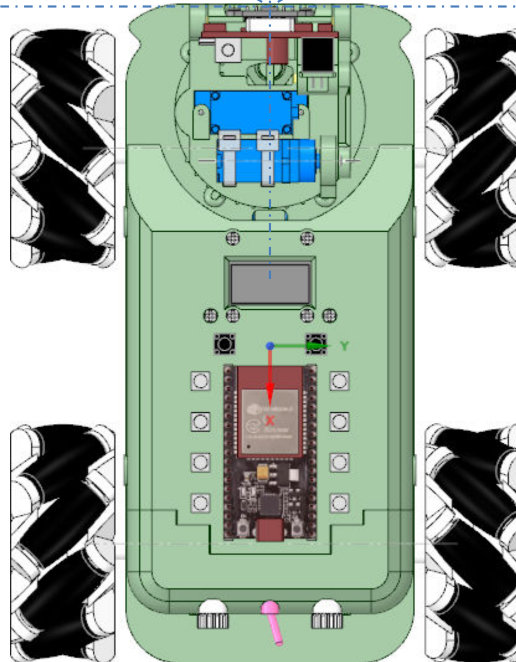
Issue: 1.3 Released: 26/11/2025 TechKnowTone

VL53L1X (LTOF) Autonomous Steering

90°
Stop!
100%
Edge
0%
Drive
0%
Wall
100%
Stop!

30°
Stop!
60%
Edge
0%
Drive
0%
Wall
60%
Stop!

150°
Stop!
60%
Edge
0%
Drive
0%
Wall
60%
Stop!

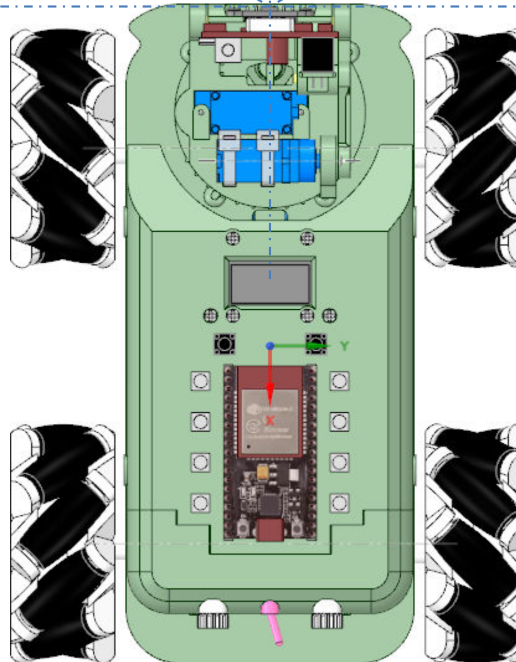


This diagram defines the autonomous steering behaviour of the PixyBot2 when using the LTOF sensor. If a wall is in the Stop! Zone then neutral turns are performed. In the Wall and Edge zones we apply proportional steering, away from the wall or edge. In the drive zones there is no steering being applied at all.

The Edge zone is only active once a ground calibration process has been successfully completed, otherwise edge detection is not possible.

Note that if someone or object is just beyond the edge, this can defeat the edge detection as the code will fail to see the transition.

VL53L1X (LTOF) Autonomous Edge Steering



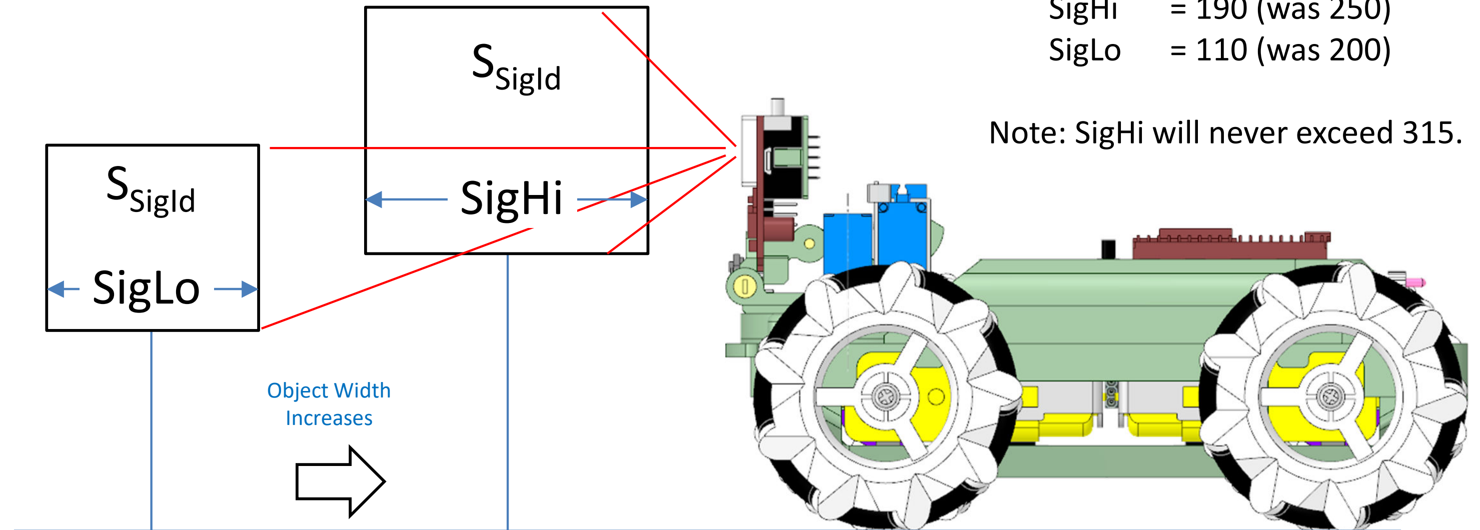
This diagram defines the steering behaviour of the PixyBot2. If a wall or edge is in the Stop! Zone then neutral turns are performed. In the Wall and Edge zones we apply proportional steering, away from the wall or edge. In the drive zones there is no steering being applied at all.

The Edge zone is only active once a ground calibration process has been successfully completed, otherwise edge detection is not possible.

Note that if someone or object is just beyond the edge, this can defeat the edge detection as the code will fail to see the transition.

Pixy2 Camera Settings

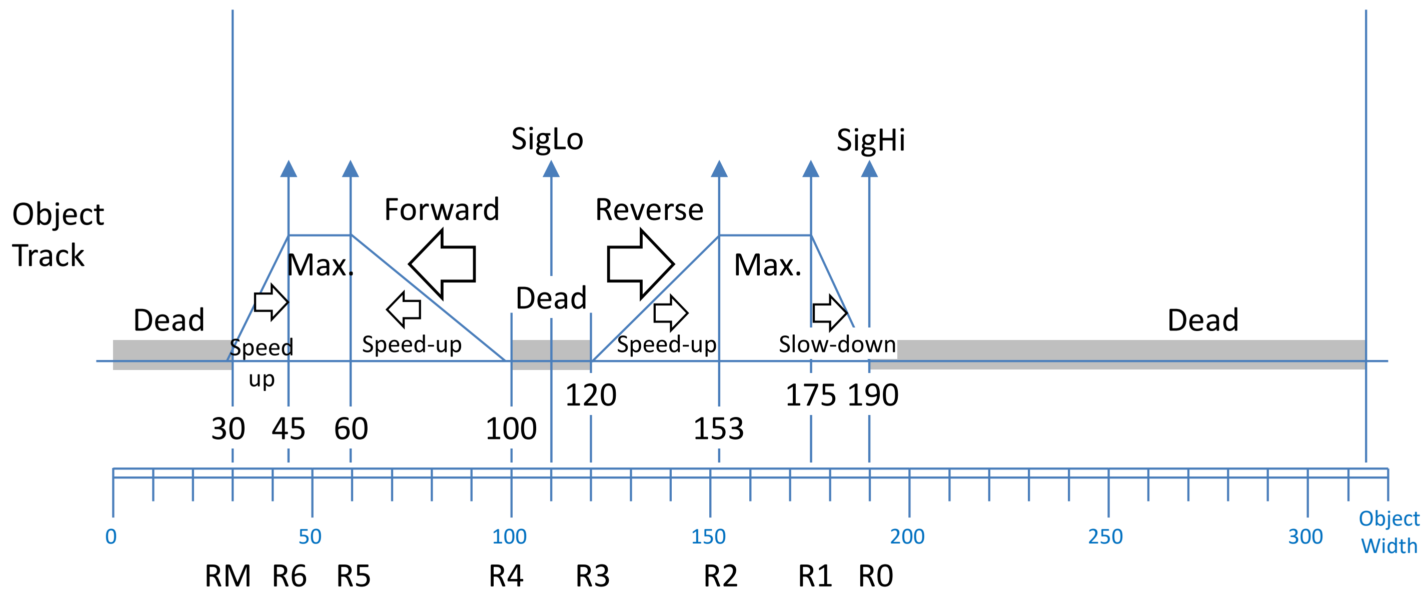
For the Pan & Tilt code in the ESP32 to lock onto a previously trained object, its width must be >= SigHi. The SigId object must then retreat to <= SigLo before any action can take place. If the object then comes close to within >= SigHi the SigId reference will be rescinded. Any trained object can take control from another by being as close as >= SigHi.



In my code I use:

SigHi = 190 (was 250)
SigLo = 110 (was 200)

Note: SigHi will never exceed 315.



When Pixy2 is tracking an object it will naturally get larger as it comes closer to the camera. Therefore I used the width of the object to set the forward and reverse drive signals for the 4x4 chassis.

The SigHi and SigLo limits are used to control the overall behaviour of the robot, whilst object width values R0 to RM control its direction and speed values.

PixyMon – Default Settings

Configure — Pixy Parameters (saved on Pixy) › Tuning

Tabs: Pixy Parameters (saved on Pixy) | PixyMon Parameters (saved on computer)

Tuning | Expert | Signature Labels | Camera | Interface | Servo

Parameter	Value
Signature 1 range	3.500000
Signature 2 range	3.500000
Signature 3 range	3.500000
Signature 4 range	3.500000
Signature 5 range	3.500000
Signature 6 range	3.500000
Signature 7 range	3.500000
Min brightness	0.200000
Camera brightness	50

OK | Cancel | Apply

Configure — Pixy Parameters (saved on Pixy) › Expert

Parameter	Value
Block filtering	3
Max tracking velocity	65
Max blocks	100
Max blocks per signature	100
Max merge dist	7
Min block area	20
LED brightness	750
Signature teach threshold	3700
Color code mode	Enabled
Default program	color connected components
Program select on power-up	☐
Debug	0

OK | Cancel | Apply

Configure — Pixy Parameters (saved on Pixy) › Signature Labels

Parameter	Value
Signature label 1	
Signature label 2	
Signature label 3	
Signature label 4	
Signature label 5	
Signature label 6	
Signature label 7	

OK | Cancel | Apply

Configure — Pixy Parameters (saved on Pixy) › Camera

Parameter	Value
Auto Exposure Correction	☑
Auto White Balance	☐
Auto White Balance on power-up	☑
Flicker avoidance	☐
Min frames per second	61

OK | Cancel | Apply

Configure — Pixy Parameters (saved on Pixy) › Interface

Parameter	Value
Data out port	I2C
I2C address	0x54
UART baudrate	19200
Pixy 1.0 compatibility mode	☐

OK | Cancel | Apply

Configure — Pixy Parameters (saved on Pixy) › Servo

Parameter	Value
S0 lower limit	-200
S0 upper limit	200
S1 lower limit	-200
S1 upper limit	200
Servo frequency	60

OK | Cancel | Apply