Introduction

This is a blueprint for a cooperative multitasking infrastructure to be implemented in the SUMO bot competition. The goal of the competition is to survive in a steel rink, bordered with white paint and be the last remaining bot after 3 minutes. The bots must startup and shutdown on a remote control command, but otherwise be completely autonomous. While most bots took the approach of chasing other bots and pushing them with brute force, our bot recognized its weaknesses and took the approach of most effectively running away and avoiding interaction.

Purpose

This code contains the various tasks and interrupts to be run during operation of our SUMO bot. It also contains a scheduling program to run the tasks cooperatively by setting timings and priorities. Shares are set and cleared according to code in hardware-based tasks, and in order to run code in strategy-based tasks. In this way, the hardware interfaces with the software.

Usage

To impliment this infrastructure, copy the list of functions, initialization code, and scheduling code into a main file with necessary cooperative multitasking files imported. The pins used correspond to pins found on the Nucleo-L476RG micro-controller and X-Nucleo-IHM04A1 DC motor driver expansion.

Testing

To determine if this code works, implement it on a SUMO bot with matching hardware and pin setup, and send a start signal to the IR reciever. Optimal testing would include a rink similar to the one used in the competition, with similar testing conditions.

Bugs and Limitations

One significant bug remaining in the program is that when the front optical sensor sees an opponent and initiates its avoidance protocol, it will sometimes run an additional time because it sees the opponent during the execution. This causes the bot to run backwards off the edge of the rink. A list of limitations can be found in the body of the report.

Location

http://wind.calpoly.edu/hg/mecha12/file/58c89a58431f/Final%20Project/main.py



motor.MotorDriver Class Reference

Public Member Functions

def __init__ (self, motor_number=1)

def set_duty_cycle (self, level)

Public Attributes

motor_number

ch1

timer channel corresponding to CPU Pin PB4. More...

ch2

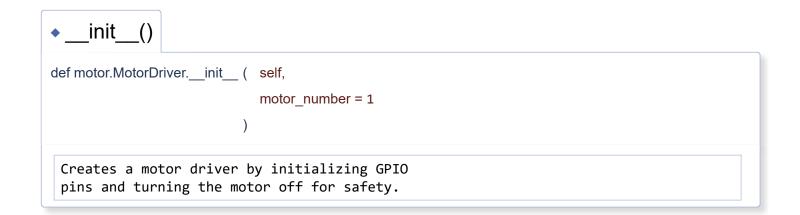
timer channel corresponding to CPU Pin PB5. More...

level

Detailed Description

This class implements a motor driver for the ME405 board.

Constructor & Destructor Documentation



Member Function Documentation

set_duty_cycle()

	def motor.	.MotorDriver.set	duty	cycle	self,
--	------------	------------------	------	-------	-------

level

This method sets the duty cycle to be sent to the motor to the given level. Positive values cause torque in one direction, negative values in the opposite direction. @param level A signed integer holding the duty cycle of the voltage sent to the motor. Saturated at range limits -100 and 100

)

Member Data Documentation

◆ ch1
motor.MotorDriver.ch1
timer channel corresponding to CPU Pin PB4. channel 1 on timer 3
channel 1 on timer 5
 ◆ ch2
motor.MotorDriver.ch2

timer channel corresponding to CPU Pin PB5.

channel 2 on timer 3

channel 2 on timer 5

The documentation for this class was generated from the following file:

• motor.py



main Namespace Reference

Functions

def irInterrupt (time1)
def EncoderInterrupt (time4)
def StartInterrupt (time8)
def IR_command_sensor ()
def SharpLeftTurn ()
def EdgeTurning ()
def MotorShutDown ()
def EdgeSensor1 ()
def EdgeSensor2 ()
def OpticalSensor1 ()
def OpticalSensor2 ()
def Strategy ()
def Garbage_Day ()
def matchTimer ()

Variables

<pre>irq_state = pyb.disable_irq ()</pre>	
time1 = pyb.Timer(1, prescaler=79, period = 65535)	
<pre>pinA8 = pyb.Pin(pyb.Pin.board.PA8, pyb.Pin.IN)</pre>	
IC	
pin	
polarity	
callback	
time4 = pyb.Timer(4, prescaler = 79, period = 65535)	
pinB7 = pyb.Pin(pyb.Pin.board.PB7, pyb.Pin.IN)	
time8 = pyb.Timer(8, prescaler = 79, period = 65535)	
<pre>pinC7 = pyb.Pin(pyb.Pin.board.PC7, pyb.Pin.IN, pyb.Pin.PULL_DOWN)</pre>	
Left = motor.MotorDriver(motor_number=1)	
Right = motor.MotorDriver(motor_number=2)	
garbage_collection	
This task corresponds with Garbage_Day() to be added to the scheduler. More	
ir_command	
This task is used with IR_command_sensor() to be added to the scheduler. More	
shutdown	

This task corresponds with MotorShutDown() to be added to the scheduler. More...

strategy

This task corresponds with Strategy() to be added to the scheduler. More...

edge1

This task corresponds with EdgeSensor1() to be added to the scheduler. More...

optics1

This task corresponds with OpticalSensor1() to be added to the scheduler. More...

edge2

This task corresponds with EdgeSensor2() to be added to the scheduler. More...

optics2

This task corresponds with OpticalSensor2() to be added to the scheduler. More...

match_timer

This task corresponds with matchTimer() to be added to the scheduler. More ...

sharpLeftTurn

This task corresponds with SharpLeftTurn() to be added to the scheduler. More...

edgeturn

This task corresponds with EdgeTurning() to be added to the scheduler. More...

ShutDownFlag

Flag that determines if the SUMO bot is in shutdown mode or not. More...

DutyCycle1 = task_share.Share ('i', thread_protect = True, name = "DC1")

Stores the current value of duty cycle for a single motor train.

DutyCycle2 = **task_share.Share** ('i', thread_protect = True, name = "DC2") Stores the current value of duty cycle for another motor train.

EncoderCounter

Stores the current count of elapsed encoder ticks. More ...

EdgeSensor1Flag

Flag that determines if an edge or line was sensed by the front sensor. More...

EdgeSensor2Flag

Flag that determines if an edge or line was sensed by the edge sensor. More...

times_up

Flag that determines if the end of the round is upon us (2min50sec) More...

sharp_left_turn

Flag that determines if the bot should make a sharp left turn. More...

edge_turn

Flag that determines if the bot should commence edge turning. More...

Side_Detection

Flag that determines if the bot detects an obstacle at its left side. More...

Front_Detection

Flag that determines if the bot detects an obstacle at its front side. More...

InterruptQueue

Queue that stores 38kHz infrared signal data collected by interrupts. More...

vcp = pyb.USB_VCP ()

Detailed Description

@file main.py
This file contains the code to run an autonomous SUMO bot

@authors Anthony Catello, Joseph Heald, Schuyler Ryan

Function Documentation

EdgeSensor1()

```
def main.EdgeSensor1 ()
```

This functions runs every 25ms and reads the edge sensor on the front of the SUMO bot and raises a flag if it detects black. Using inverted logic, action is taken when the flag is cleared-indicating a white line or edge.

EdgeSensor2()

def main.EdgeSensor2 ()

This functions runs every 25ms and reads the edge sensor on the right side of the SUMO bot and raises a flag if it detects black. Using inverted logic, action is taken when the flag is cleared-indicating a white line or edge.

EdgeTurning()

def main.EdgeTurning ()

This function runs every 25ms and makes the SUMO bot maintain a close orbit near the white edges of the rink by turning slightly.

EncoderInterrupt()

def main.EncoderInterrupt (time4)

This function runs when an interrupt is triggered corresponding to timer 4. Such an interrupt is triggered when the signal on the motor encoder is disrupted by motor rotation.

Garbage_Day()

def main.Garbage_Day ()

This function runs every 2000ms and clears out memory every so often, no big deal.

IR_command_sensor()

```
def main.IR_command_sensor ()
```

This function is run every 100ms and processes any data that has been collected in the InterruptQueue from irInterrupt().

irInterrupt()

def main.irInterrupt (time1)

This function runs when an interupt is recieved corresponding to timer 1. Such an interupt is recieved when a 34kHz infrared signal is read from the IR reciever located on the SUMO bot.

matchTimer()

def main.matchTimer ()

This function runs every 1000ms and keeps track of how much time has elapsed since the start of the match



def main.MotorShutDown ()

This function runs every 100ms and turns the motors off on the SUMO bot. Other functions remain powered, but the bot stops moving of its own accord.

OpticalSensor1()

def main.OpticalSensor1 ()

This function runs every 25ms and reads the optical sensor on the front of the SUMO bot and raises a flag if it reads a distance less than 8".

OpticalSensor2()

def main.OpticalSensor2 ()

```
This function runs every 25ms and reads the optical sensor on the side of the SUMO bot and raises a flag if it reads a distance less than 8".
```

SharpLeftTurn()

def main.SharpLeftTurn ()

This function is run every 25ms and makes the SUMO bot take a sharp turn approximately 90 degrees counterclockwise.

StartInterrupt()

def main.StartInterrupt (time8)

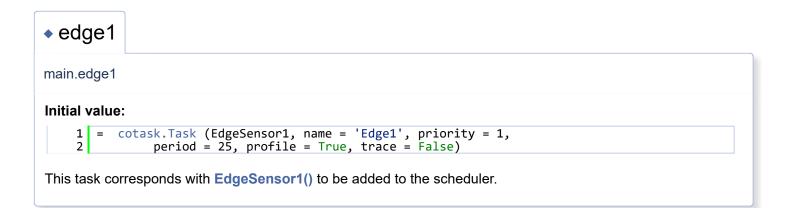
This function runs when an interrupt is triggered corresponding to timer 8. Such an interrupt is recieved when start-up button is pressed.

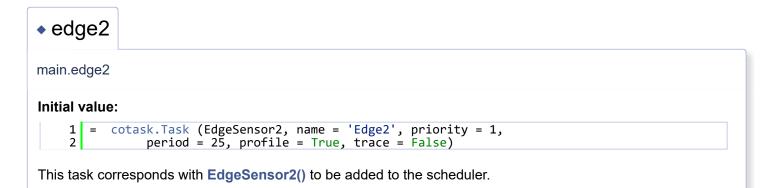
Strategy()

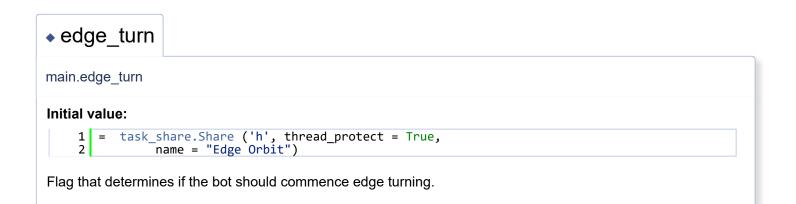
def main.Strategy	()
-------------------	---	---

This function runs every 25ms and is essentially the mastermind, or hub. It runs time-based procedures and sets flags for event based procedures.

Variable Documentation







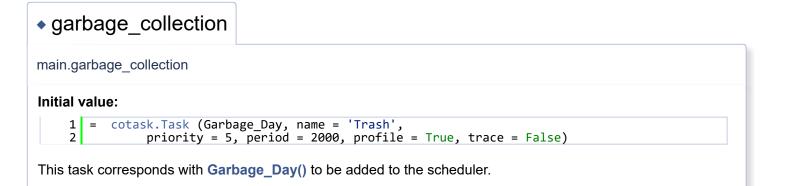
main.EdgeSensor1Flag	
Initial value:	
<pre>1 = task_share.Share ('h', thread_protect = True, 2 name = "SideEdge")</pre>	
Flag that determines if an edge or line was sensed by the front sensor.	

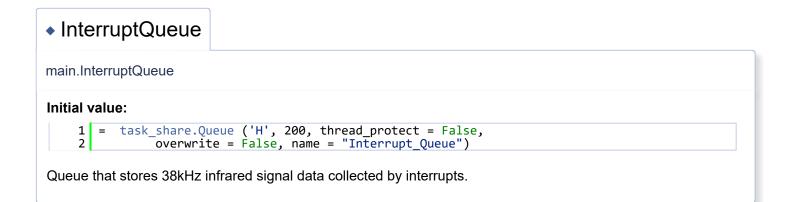
◆ EdgeSensor2Flag	
main.EdgeSensor2Flag	
Initial value: 1 2 task_share.Share ('h', thread_protect = True, name = "FrontEdge") Flag that determines if an edge or line was sensed by the edge sensor.	
 ◆ edgeturn 	
main.edgeturn	
Initial value: 1 2 period = 25, profile = True, trace = False) This task corresponds with EdgeTurning() to be added to the scheduler.	

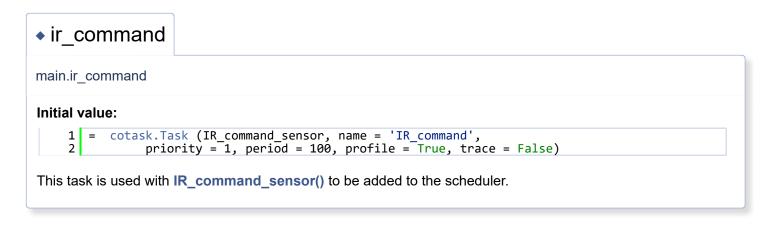
EncoderCounter main.EncoderCounter Initial value: 1 = task_share.Share ('1', thread_protect = True, 2 name = "Encoder 1") Stores the current count of elapsed encoder ticks.

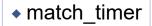
Front_Detection

main.Front_Detection	
Initial value:	
<pre>1 = task_share.Share ('h', thread_protect = True, 2 name = "front_detection")</pre>	
Flag that determines if the bot detects an obstacle at its front side.	



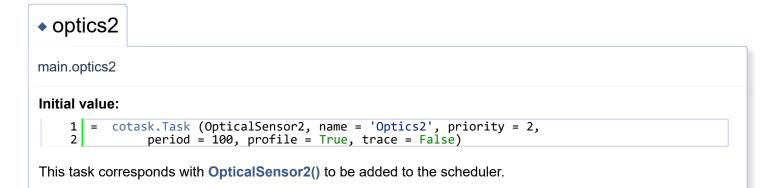


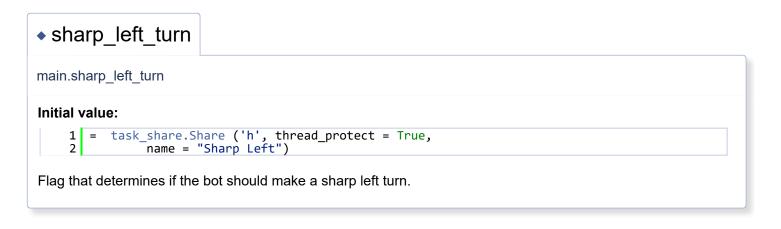




main.match_timer
Initial value:
<pre>1 = cotask.Task (matchTimer, name = "match_timer", priority = 4, 2 period = 1000, profile = True, trace = False)</pre>
This task corresponds with matchTimer() to be added to the scheduler.

 ◆ optics1
main.optics1
<pre>Initial value: 1 = cotask.Task (OpticalSensor1, name = 'Optics1', priority = 2, 2</pre>



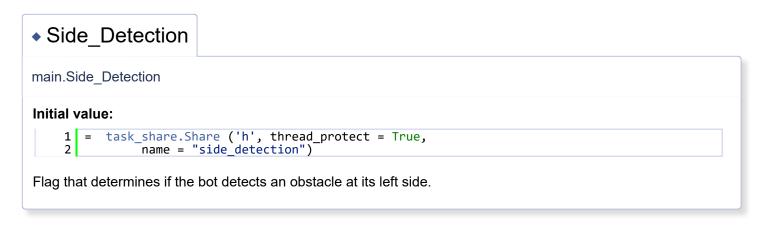


main.sharpLeftTurn
Initial value:
<pre>1 = cotask.Task (SharpLeftTurn, name = "sharp_left_turn", 2 priority = 2, period = 25, profile = True, trace = False)</pre>
This task corresponds with SharpLeftTurn() to be added to the scheduler.

 shutdown 	
main.shutdown	
•	<pre>c.Task (MotorShutDown, name = 'Shutdown', priority = 1, beriod = 100, profile = True, trace = False) onds with MotorShutDown() to be added to the scheduler.</pre>

ShutDownFlag

main.ShutDownFlag
Initial value:
1 = task_share.Share ('h', thread_protect = True, 2 name = "ShutDown")
Flag that determines if the SUMO bot is in shutdown mode or not.





main.strategy
Initial value:
1 = cotask.Task (Strategy, name = 'strategy', priority = 1, 2 period = 25, profile = True, trace = False)
This task corresponds with Strategy() to be added to the scheduler.

◆ times_up	
main.times_up	
<pre>Initial value: 1 = task_share.Share ('h', thread_protect = True, 2</pre>	

