Syringenator

Generated by Doxygen 1.8.16

1 README

University of Washington

TCES 460

Winter 2019

1.1 Development Team Vulcan

Authors

Name	Email
Alex Boyle	boylea4@uw.edu
Ammon Dodson	ammon0@uw.edu
Alex Marlow	alexmarlow117@gmail.com
Jake McKenzie	jake314@uw.edu
Brooke Stevenson	brooks04@uw.edu

Copyright

1.2 Using Git

Git is a command-line tool for managing source code. Github is an on-line service that provides git remotes. A git remote is a remote copy of a git repository. Multiple people work in the same repository through the use of a single remote. The trick is to manage version conflicts intelligently.

Each team member should periodically merge master into their own branch to ensure that we are synced up. The master branch should only ever have merge commits and working code. I will try to enforce this with Github so that we don't make a mess. –ABD

1.2.1 Work in Your Own Branch

Each team member should create their own branch to work in. You may make as many branches as you like, just make sure you have one. You can create branches on the command line with:

```
$ git branch <branch-name>
```

To switch to your branch do:

```
$ git checkout <branch-name>
```

1.2.2 Commit Your Work

Commits are a permanent record of your work. They should be as small and purpose-driven as possible. Think: "can I write a couple lines that explains what I did?" To check for uncommitted changes, or check your status in general do:

```
$ git status
On branch ammon
Your branch is up-to-date with 'github/ammon'.  <- this is the remote
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout - <file>..." to discard changes in working directory)
    modified:   README.md
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    docs/autotoc_md6.html
    latex/autotoc_md6.tex
no changes added to commit (use "git add" and/or "git commit -a")
```

You make a commit in two steps: first you stage the changed files that you want to include in this next commit.

```
$ git add <filename> <anotherfile>
```

Once you have staged a bunch of changes you can check your status again:

```
$ git status
On branch ammon
Your branch is up-to-date with 'github/ammon'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    modified:   README.md
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout - <file>..." to discard changes in working directory)
    modified:   Makefile
    deleted:    refman.pdf
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    docs/autotoc_md7.html
    latex/autotoc_md7.tex
```

Once you are satisfied with what is currently staged you finish the commit by doing:

```
$ git commit
```

Git will open a text editor where you can describe what the changes are. Make this a meaningful message since it will be the only thing that distinguishes this commit from hundreds of others.

1.2.3 Merge All the Latest Changes

The magic of git is being able to merge conflicting changes. Before you share your changes (pushing), you must pull the latest changes and merge them with yours. First pull the master branch:

```
$ git pull origin master
```

You will need to enter your password and git will tell you if there have been any changes. Then you will merge:

```
$ git merge master
```

Git will attempt to merge the master branch into yours. If there are any conflicts it will tell you. Git will rewrite your files to include both versions of the conflicting code. To see which files are in conflict do:

```
$ git status
```

You have to open those files, find, and fix the conflicting versions. Once you think you are done, rebuild and test all the code. Look for any new errors and fix them. Once you are satisfied that the merge has been completed successfully add and commit your changes as usual.

1.2.4 Push Your Branch

Pushing your work to the remote allows everyone else to see it. You should merge master before pushing. To push do:

```
$ git push origin <your-branch>
```

1.3 HypoRobot Assignment

If you've been paying any attention at all to current events you know that a major plague has descended on cities and counties throughout the country in the form of used and discarded hypodermic needles. Countless hours are spent cleaning up this mess. For instance, some schools are forced, for safety reasons, to send staff out to scour the playgrounds prior to children showing up.

Your task this quarter will be to design an autonomous robot that can help automate the arduous and sometimes dangerous job of spotting, retrieving, and disposing of hypodermic syringes.

Your robot will be a prototype, not a fully functional disposal robot, but it will have important technical features necessary on such a robot.

A second point is that we will be dealing with industrial (i.e. dull) syringes. These are typically used to disburse such things glue or solvents. They are commonly used in our labs to glue acrylic parts together. Anyone in the lab with a sharp needle will be immediately disqualified. Even so, if you would rather not design and test with any syringe, you may, with my written permission, use a ballpoint pen, a #2 pencil or a similar object of your choosing.

All testing will be done indoors on a flat surface.

–Robert Gutmann, Ph.D.

2 Requirements

Author

Robert Gutmann, Ph.D.

2.1 Terminology

The following terms are used in this specification:

- The term "autonomous", in this case, means that no commands can be transmitted to your robot from any outside agency (especially from a human or computer or other controller) and all sensors used in the contest must be physically attached to your robot. No wired connections are allowed between any outside agency and your robot.
- The term "course" refers to the area in which the contest takes place.
- The term "tape line" refers to an oval of white tape that runs from a start point around the oval, back to the start point (which is now the finish point). All targets will be placed outside of the oval.
- The term "target" refers to the object you are required to pick up and dispose of (syringe or, alternatively, a pen or pencil).

- The term “decoy” refers an object on the course that is not a target. A decoy will be less than 2 cm tall.
- The term “obstacle” refers to an object on the course that your robot must avoid running into. An obstacle will be at least 15 cm tall. A typical obstacle would be a cardboard box.
- The term “finish the course” will mean that your robot traverses the oval at least once. Note: Your robot will have to leave the tape line to pick up targets, but it should eventually either find another target or return to the tape line. The tape line is your navigation aid.
- The term “contact a target” will mean to touch a target with your pick-up mechanism in such a way as to move it. Note: moving a target with a robot wheel or track does not count as a contact.
- The term “participate” will mean that you either finish the course or contact a target.
- The term “acquire a target” means your robot has reported to its data logger that it has identified a target and reports an accurate position for that target. The term “acquire a decoy” means your robot has reported to its data logger that it has acquired a target that turns out to be a decoy.
- The term “pick up a target” refers to your robot picking up a target off the course surface.
- The term “dispose of a target” refers to your robot placing the target in container on your robot.
- A robot is “stationary” if its wheels are not rotating and its arm is not rotating about its vertical axis.

2.2 Rules of the Game

- You will be given two test runs, one per day over two class periods. The dates will be firmly established by midterm time.
- All tests will be conducted indoors.
- A somewhat different course may be laid out each day. The layout will consist of:
  - A tape line; this will serve as your navigation maker. Since we will be indoors, we won’t have GPS; the tape line will serve as your navigation reference.
  - A number of targets will be placed within 1 meter of the tape line; you will have to leave the tape line to pick up your targets.
  - A number of decoys will be placed within 1 meter of the tape line.
  - A number of obstacles will be placed on the course. If you exactly follow the tape line you will not run into an obstacle; however, you may have to avoid obstacles as you maneuver away from the tape line to pick up targets.
- No human will be allowed on the course during a test run.
- Your robot must be autonomous.
- All test runs will be video ‘taped.’
- The goal is to maximize your score according to the algorithm discussed below. The maximum score you achieve for any one day over the two days will be your final score.
- The scores for the entire class will be rank-ordered.
- You will be allowed ten minutes on the course for each test run. This will be strictly timed.
- Robot
  - You will be provided with

    - A basic robot chassis
    - Two motors with encoders and wheels
    - Two motor controllers (H-bridges)
    - A robotic arm
    - A battery pack with a power distribution unit
    - Distance sensors.
    - Line sensors
    - Data logger with SD card
  - You do not have to use this robot chassis or arm
  - You will need to supply your own processor(s)
  - You will need to supply your own cameras(s) and cables.
  - You may acquire additional mechanical or electronic parts for your robot.
  - If you plan to spend any money on your robot, you must get permission from me in writing first.
  - Your group has a strict budget of $300, including any parts that you have already acquired and use on your robot (e.g., an Arduino).
- Rule 8 applies. Rule 8 comes from the official rules for the annual Race to Alaska (see https://r2ak.com/official-rules/). Rule 8 states, and I quote: If we decide it's necessary to consult a lawyer to figure out if you are disqualified or not, you are automatically disqualified. Play by the rules and live up to the spirit of the race. If you get cute and push the boundaries, we'll bring down the hammer.

3 Todo List

File Syringenator.py

TODO: how do we initialize the robot run? a button press? –ABD

Member Syringenator.returnToLine ()

TODO: do we need to check that we actually returned? how do we recover if dead reckoning fails? –ABD

4 Namespace Index

4.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

5 Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6 File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

7 Namespace Documentation

7.1 Syringenator Namespace Reference

The top-level Pi program.

Classes

- class Target

  A class to contain everything we know about an aquired target.

Functions

- def log (arg)

  Record system events for later analysis.
- def arduinoSend (bytes)

  Send serial data to the arduino.
- def arduinoReceive ()

  Wait some fixed time for the arduino to send one or more bytes.
- def scan ()

A routine to take a picture and report back the closest target The Computer vision routine must be able to handle multiple targets in the image.

- def moveCloser (t)

  Move the robot closer to the given target.
- def pickUp (t)

  Attempt to pickup and dispose the target.
- def returnToLine ()

  signl the arduino to return to the line.
- def lineFollow ()

  Follow the line.
- def canBePicked (t)

  A routine to determine if the target is in position to be picked up.

Variables

- bool onTheLine = True

  boolean indicating whether we are on the line
- target = None

  The currently aquired target.

7.1.1 Detailed Description

The top-level Pi program.

7.1.2 Function Documentation

7.1.2.1 arduinoReceive()

```
def Syringenator.arduinoReceive ( )
```

Wait some fixed time for the arduino to send one or more bytes.

Returns

a list of bytes

7.1.2.2 arduinoSend()

```
def Syringenator.arduinoSend (
              bytes )
```

Send serial data to the arduino.

Parameters

bytes	one or more bytes of data to send to the arduino

Returns

None

7.1.2.3 canBePicked()

```
def Syringenator.canBePicked (
              t )
```

A routine to determine if the target is in position to be picked up.

Calculates whether the center of the target bounding box is in the pickup area.

Returns

a boolean

7.1.2.4 lineFollow()

```
def Syringenator.lineFollow ( )
```

Follow the line.

this routine simply signals the arduino to execute its lineFollow() routine

Returns

None

7.1.2.5 log()

```
def Syringenator.log (
              arg )
```

Record system events for later analysis.

Returns

None

7.1.2.6 moveCloser()

```
def Syringenator.moveCloser (
              t )
```

Move the robot closer to the given target.

The moveCloser() routine attempts to aproach the target by relatively small increments. Because the move routines may be interrupted by the obstacle avoidance ISRs and the risk of jambing the wheels etc. we cannot expect to be able to approch successfully on the first try. Hence moveCloser() should only move a relatively short distance before exiting to allow another loop through the scan cycle.

Should we spend effort trying to avoid running over decoys here?

This routine should check for ARDUINO_STATUS_OBSTACLE. then what?

This routine is likely where we will have the most issues. –ABD

Parameters

t	a Target object containing the location of the target to be approched

Returns

None

7.1.2.7 pickUp()

```
def Syringenator.pickUp (
              t )
```

Attempt to pickup and dispose the target.

This routine must determine orientation of the target. If this is not done by some OpenCV magic we can attempt it here using the raw image data and the bounding box.

Divide the longer dimension of the bounding box by some constant divisor. Scan along each of those raster lines twice. On the first pass calculate an average brightness (RGB values can be summed). The second pass will pick out points of greatest brightness. Find the centers of clustered bright pixeles. We now have a set of points in cartesian space. Have Jake find the slope of the line of best fit.

The center can be estimated as the center of the bounding box, or the center of the points, the mean of both, etc.

Once the values for x, y, and m have been determined they will have to pass through a calibration transform to determine the arm a, r, o values. –ABD

Parameters

t	a Target object containing the raw bitmap data

Returns

None

7.1.2.8 returnToLine()

```
def Syringenator.returnToLine ( )
```

signl the arduino to return to the line.

Todo TODO: do we need to check that we actually returned? how do we recover if dead reckoning fails? –ABD

Returns

None

7.1.2.9 scan()

```
def Syringenator.scan ( )
```

A routine to take a picture and report back the closest target The Computer vision routine must be able to handle multiple targets in the image.

It would be best if all targets are reported. Then this routine will determine the closest one to pursue. –ABD

Returns

a target object

8 Class Documentation

8.1 Syringenator.Target Class Reference

A class to contain everything we know about an aquired target.

8.1.1 Detailed Description

A class to contain everything we know about an aquired target.

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

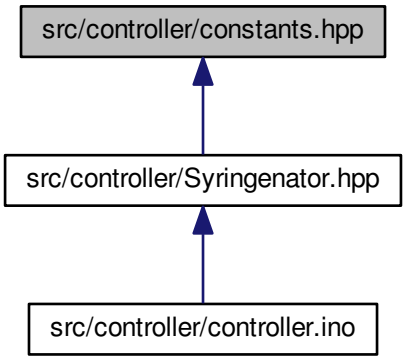
- src/pi/Syringenator.py

9 File Documentation

9.1 src/controller/constants.hpp File Reference

Constants shared across the whole system.

This graph shows which files directly or indirectly include this file:



Macros

- #define ARM_AZIMUTH_MIN 0

  The minimum azimuth byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define ARM_AZIMUTH_MAX 0

  The maximum azimuth byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define ARM_RANGE_MIN 0

  The minimum range byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define ARM_RANGE_MAX 0

  The maximum range byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define ARM_ORIENT_MIN 0

  The minimum orientation byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define ARM_ORIENT_MAX 0

  The maximum orientation byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- #define PICKUP_X_MIN 0

  The minimum target center x-value that allows a pickup.
- #define PICKUP_X_MAX 0

  The maximum target center x-value that allows a pickup.
- #define PICKUP_Y_MIN 0

  The minimum target center y-value that allows a pickup.
- #define PICKUP_Y_MAX 0

The maximum target center y-value that allows a pickup.

- #define ARDUINO_NULL 0x00

  A place holder for troubleshooting etc.
- #define ARDUINO_STATUS_ACK 0x01

  If the arduino needs to acknowledge something.
- #define ARDUINO_STATUS_READY 0x02

  If the arduino needs to indicate it is ready.
- #define ARDUINO_STATUS_PICK_FAIL 0x03

  Report that the pick failed.
- #define ARDUINO_STATUS_PICK_SUCCESS 0x04

  Report that the pick succeded.
- #define ARDUINO_STATUS_ARM_FAULT 0x05

  Report a general arm failure.
- #define ARDUINO_STATUS_OBSTACLE 0x06

  Report an obstacle detected.
- #define ARDUINO_ROTATE 0x10

  serial command the arduino to rotate the robot, followed by one signed byte indicating magnitude and direction
- #define ARDUINO_MOVE 0x11

  serial command the arduino to advance the robot, followed by one signed byte indicating magnitude and direction
- #define ARDUINO_LINE_FOLLOW 0x12

  serial command the arduino to follow the line
- #define ARDUINO_ARM_PARK 0x20

  serial command the arduino to call the park action sequence
- #define ARDUINO_ARM_DISPOSE 0x21

  serial command the arduino to call the dispose action sequence
- #define ARDUINO_ARM_PICKUP 0x22

  serial command the arduino to attempt a pick, followed by three bytes: azimuth, range, and orientation
- #define PORT_MOTOR_FWD None

  Arduino pin for port motor forward.
- #define PORT_MOTOR_REV None

  Arduino pin for port motor reverse.
- #define STBD_MOTOR_FWD None

  Arduino pin for starboard motor forward.
- #define STBD_MOTOR_REV None

  Arduino pin for starboard motor reverse.
- #define PORT_LINE_SENSE None

  Arduino pin for the port line sensor.
- #define STBD_LINE_SENSE None

  Arduino pin for the starboard line sensor.
- #define PORT_FWD_OBSTACLE None

  Arduino pin for the port forward obstacle sensor.
- #define PORT_AFT_OBSTACLE None

  Arduino pin for the port aft obstacle sensor.
- #define STBD_FWD_OBSTACLE None

  Arduino pin for the starboard forward obstacle sensor.
- #define STBD_AFT_OBSTACLE None

  Arduino pin for the starboard aft obstacle sensor.
- #define ARM_CONTROL None

  Arduino pin for communication with the xArm.

9.1.1 Detailed Description

Constants shared across the whole system.

Includes constants used by both the arduino sketch and the the python script. The format of `constants.in` is three whitespace sparated columns:

[NAME] [value] [comments]

Any changes must be made in `constants.in` and followed by running:

```
make constants
```

–ABD

Copyright

> Copyright © 2019 by the authors. All rights reserved.

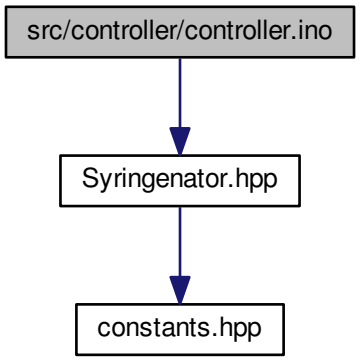
This file has been autogenerated, CHANGES MADE HERE WILL NOT PERSIST

9.2 src/controller/controller.ino File Reference

The Arduino sketch.

```
#include "Syringenator.hpp"
```

Include dependency graph for controller.ino:



Functions

- void setup ()

  Setup code here.
- void loop ()

  Controller loop code here.

9.2.1 Detailed Description

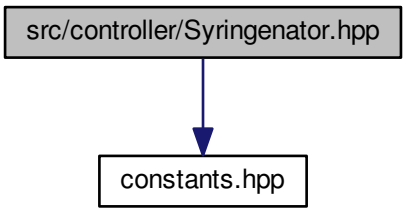
The Arduino sketch.

9.3 src/controller/Syringenator.hpp File Reference

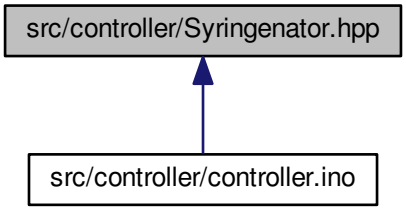
Arduino controller code –ABD.

```
#include "constants.hpp"
```

Include dependency graph for Syringenator.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void lineDetector_ISR (void)

  A function to respond to a line detector being triggered.
- void obstacleDetector_ISR (void)

  A function to respond to a detected obstacle while under locamotion.
- void motorEncoder_ISR (void)

  Motor encoder ISR.
- void serialCommunication_ISR (void)

  A function to handle incomming communication from the pi.
- void moveRotate (int ticks)

  Rotate the robot around central axis rotate by running both motors at the same speed in opposite directions.
- void moveStraight (int ticks)

  Move the robot forward or reverse.
- void moveLineFollow (void)

  Routine to follow the guide-line for some fixed interval.
- void armPark (void)

  Move the arm to its parking position.
- void armDispose (void)

  Routine to dispose of a syringe once it has been picked.
- bool armPick (byte azimuth, byte range, byte orientation)

  Routine to attempt target pickup.

9.3.1 Detailed Description

Arduino controller code –ABD.

Copyright

9.3.2 Function Documentation

9.3.2.1 armPark()

```
void armPark (
            void  )
```

Move the arm to its parking position.

The parking position needs to leave a clear view of the pickup area, but also should move the center of gravity as far forward as possible to reduce drive wheel slippage.

9.3.2.2 armPick()

```
bool armPick (
            byte azimuth,
            byte range,
            byte orientation )
```

Routine to attempt target pickup.

This routine should attempt to close the claw completely and detect if an object as actually been grabbed. parameters should be bytes because they will have to be transmitted over serial from the pi. Ranges on these values TBD as convenient for the arm software, but must be recorded in the system constants file. –ABD

Parameters

azimuth	arm azimuth value
range	distance to the target
orientation	rotation of the target

Returns

true on successful pick, false otherwise.

9.3.2.3 lineDetector_ISR()

```
void lineDetector_ISR (
             void  )
```

A function to respond to a line detector being triggered.

The line detectors are mounted forward and inboard of the wheels. This function needs to reorient the robot to clear the sensor, but also to prevent the line from being hit again.

The simplest way to do this is to rotate the opposite wheel forward until the sensor clears. Because the sensor is forward of the wheel it will rotate away from the line as the opposite wheel moves forward. This should work as long as the curvature of the line is not too great.

This may need to be two routines, one for each sensor –ABD

9.3.2.4 motorEncoder_ISR()

```
void motorEncoder_ISR (
             void  )
```

Motor encoder ISR.

9.3.2.5 moveLineFollow()

```
void moveLineFollow (
             void  )
```

Routine to follow the guide-line for some fixed interval.

This function assumes that we are already over the line

9.3.2.6 moveRotate()

```
void moveRotate (
             int ticks )
```

Rotate the robot around central axis rotate by running both motors at the same speed in opposite directions.

Parameters

ticks	sign indicates direction of rotation: positive is rotation to the right. magnitude indicates the number of encoder ticks on each motor.

9.3.2.7 moveStraight()

```
void moveStraight (
            int ticks )
```

Move the robot forward or reverse.

Parameters

ticks	number of encoder ticks to move. Sign indicates direction: positive is forward.

9.3.2.8 obstacleDetector_ISR()

```
void obstacleDetector_ISR (
            void  )
```

A function to respond to a detected obstacle while under locamotion.

There may be two cases to handle: whether we are line following, or aproaching. If we are line following we need to ensure that we don't lose the line while avoiding the obstacle.

This may need to be multiple routines, one for each sensor –ABD

9.4 src/pi/constants.py File Reference

Constants shared across the whole system.

Variables

- int constants.ARM_AZIMUTH_MIN = 0

  The minimum azimuth byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- int constants.ARM_AZIMUTH_MAX = 0

  The maximum azimuth byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- int constants.ARM_RANGE_MIN = 0

  The minimum range byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- int constants.ARM_RANGE_MAX = 0

  The maximum range byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.

- int constants.ARM_ORIENT_MIN = 0

  The minimum orientation byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- int constants.ARM_ORIENT_MAX = 0

  The maximum orientation byte value that can be passed to the arduino with ARDUINO_ARM_PICKUP.
- int constants.PICKUP_X_MIN = 0

  The minimum target center x-value that allows a pickup.
- int constants.PICKUP_X_MAX = 0

  The maximum target center x-value that allows a pickup.
- int constants.PICKUP_Y_MIN = 0

  The minimum target center y-value that allows a pickup.
- int constants.PICKUP_Y_MAX = 0

  The maximum target center y-value that allows a pickup.
- int constants.ARDUINO_NULL = 0x00

  A place holder for troubleshooting etc.
- int constants.ARDUINO_STATUS_ACK = 0x01

  If the arduino needs to acknowledge something.
- int constants.ARDUINO_STATUS_READY = 0x02

  If the arduino needs to indicate it is ready.
- int constants.ARDUINO_STATUS_PICK_FAIL = 0x03

  Report that the pick failed.
- int constants.ARDUINO_STATUS_PICK_SUCCESS = 0x04

  Report that the pick succeded.
- int constants.ARDUINO_STATUS_ARM_FAULT = 0x05

  Report a general arm failure.
- int constants.ARDUINO_STATUS_OBSTACLE = 0x06

  Report an obstacle detected.
- int constants.ARDUINO_ROTATE = 0x10

  serial command the arduino to rotate the robot, followed by one signed byte indicating magnitude and direction
- int constants.ARDUINO_MOVE = 0x11

  serial command the arduino to advance the robot, followed by one signed byte indicating magnitude and direction
- int constants.ARDUINO_LINE_FOLLOW = 0x12

  serial command the arduino to follow the line
- int constants.ARDUINO_ARM_PARK = 0x20

  serial command the arduino to call the park action sequence
- int constants.ARDUINO_ARM_DISPOSE = 0x21

  serial command the arduino to call the dispose action sequence
- int constants.ARDUINO_ARM_PICKUP = 0x22

  serial command the arduino to attempt a pick, followed by three bytes: azimuth, range, and orientation
- constants.PORT_MOTOR_FWD = None

  Arduino pin for port motor forward.
- constants.PORT_MOTOR_REV = None

  Arduino pin for port motor reverse.
- constants.STBD_MOTOR_FWD = None

  Arduino pin for starboard motor forward.
- constants.STBD_MOTOR_REV = None

  Arduino pin for starboard motor reverse.
- constants.PORT_LINE_SENSE = None

Arduino pin for the port line sensor.

- constants.STBD_LINE_SENSE = None

  Arduino pin for the starboard line sensor.

- constants.PORT_FWD_OBSTACLE = None

  Arduino pin for the port forward obstacle sensor.

- constants.PORT_AFT_OBSTACLE = None

  Arduino pin for the port aft obstacle sensor.

- constants.STBD_FWD_OBSTACLE = None

  Arduino pin for the starboard forward obstacle sensor.

- constants.STBD_AFT_OBSTACLE = None

  Arduino pin for the starboard aft obstacle sensor.

- constants.ARM_CONTROL = None

  Arduino pin for communication with the xArm.

9.4.1 Detailed Description

Constants shared across the whole system.

Includes constants used by both the arduino sketch and the the python script. The format of `constants.in` is three whitespace sparated columns:

[NAME] [value] [comments]

Any changes must be made in `constants.in` and followed by running:

```
make constants
```

–ABD

Copyright

Copyright © 2019 by the authors. All rights reserved.

This file has been autogenerated, CHANGES MADE HERE WILL NOT PERSIST

9.5 src/pi/Syringenator.py File Reference

This is the main control script.

Classes

- class Syringenator.Target

  A class to contain everything we know about an aquired target.

Namespaces

- Syringenator

  The top-level Pi program.

Functions

- def Syringenator.log (arg)

  Record system events for later analysis.
- def Syringenator.arduinoSend (bytes)

  Send serial data to the arduino.
- def Syringenator.arduinoReceive ()

  Wait some fixed time for the arduino to send one or more bytes.
- def Syringenator.scan ()

  A routine to take a picture and report back the closest target The Computer vision routine must be able to handle multiple targets in the image.
- def Syringenator.moveCloser (t)

  Move the robot closer to the given target.
- def Syringenator.pickUp (t)

  Attempt to pickup and dispose the target.
- def Syringenator.returnToLine ()

  signl the arduino to return to the line.
- def Syringenator.lineFollow ()

  Follow the line.
- def Syringenator.canBePicked (t)

  A routine to determine if the target is in position to be picked up.

Variables

- bool Syringenator.onTheLine = True

  boolean indicating whether we are on the line
- Syringenator.target = None

  The currently aquired target.

9.5.1 Detailed Description

This is the main control script.

It will run on the Raspberry Pi and direct all robot operations.

Todo TODO: how do we initialize the robot run? a button press? –ABD

Copyright

Index