RIG InMoov Project

Generated by Doxygen 1.8.9.1

Sat Aug 1 2015 10:43:59

Contents

Chapter 1

Introduction

This is the official documentation for the Conestoga Robotics Innovation Group's InMoov project. It is not meant to be read linearly or all at once. Rather, readers should skim relevant pages and skip information that requires more context. More in depth information is provided for those that wish to modify the source code. Examples are provided when possible for quick reference and experimentation. Note that colored phrases are links for both html and pdf. This project has a git repository located `here`.

1.1 Dependencies

The project currently depends on `Python` for an example script, `Arduino`, `Doxygen` (if you wish to remake the documentation) and `Make` (if you wish to build the project).

1.2 Quick Start

To quickly upload to the right hand, do the following. Copy the **arduino/servo** folder to your Arduino sketch folder. Copy **servo/settings/rhand.h** to **servo/settings.h** in the sketch directory. Copy **arduino/lib/serial.ino** to **servo/serial.ino** in the sketch directory. Open the Arduino software. Select your board and the port it is connected to. Open your sketch, and hit upload. Instructions for the Arduino software are located `on its website`. Other Arduino files on the bot can be built in a similar way, or the **make** system command can be used.

To transmit commands to the board, run this from the command line.

```
1 python -i example/rhand.py
```

You can now run commands defined in rhand.

The python script needs the pyserial library. Its website is located `here.`

Call this to connect to the board (replace COM0 with the name of the port you set in Arduino).

```
1 connect ('COM0')
```

To demo some movements, run this.

```
1 demo ()
```

Note that some movements require input before continuing (press enter).

Read the rhand documentation for more. Read the Examples section to understand how it formulates commands.

1.3 Installation

You can build this documentation with the command

```
1 make documentation
```

from the root directory.

To upload the right hand servo code to the connected Arduino, run

```
1 make servo.ino SETTINGS=rhand ARDUINO_DO=--upload BOARD=nano
```

Assuming the board is the arduino nano.

1.4 Build System

Arduino Directory Structure

The Arduino IDE abstracts many tools for working with Arduino hardware. We choose to use it because of its familiarity and cross system compatibility. However, the Arduino IDE isn't made for command line compilation, and replicating its process requires complex tools not appropriate for this project.

To share code among certain boards, general code is kept in the **arduino**/**sketch** subdirectories. Inside each subdirectory is an .ino file of the same name, which is the source for the sketch. There is also a **settings** folder in which various headers are kept. These headers are to be moved to **settings.h** inside the sketch folder. For example, code to be shared among servos is kept in **arduino/sketch/servo/servo.ino**, the configuration for the right hand is kept in **arduino/sketch/servo/settings/rhand.h**, and you would move it to **arduino**/**sketch**/**servo**/**settings.h** to use it.

Some sketches require shared code that use Arduino libraries, which means that they need to be contained in .ino files. Arduino includes any .ino files inside a sketch subdirectory. We keep such files in **arduino/lib**. For example, servo boards require serial communication, so we would move **arduino/lib/serial.ino** to the **arduino**/**sketch**/**servo** directory.

The reason that shared code among the same board types is treated differently than shared code among different board types is to allow us to keep the settings files short, and also to use them elsewhere in the project that is not necessarily Arduino code.

This is all automated by the build system. If you wish to use the Arduino IDE, either link the appropriate files to your Arduino sketch directory or copy them to the sketch directory and write the changes back later.

1.5 Arduino Protocol

The Arduinos controlling the various parts of the bot communicate over their serial ports. All information is sent and recieved using 8 bit unsigned raw integers instead of text. This allows extremely fast and simple code, but limits programs to 255 signals, identifiers, and values.

However, the Arduino boards only have one function; to pass commands to their servos.

Servo ID

Unless otherwise instructed, the Arduino accepts two values at a time. First, it accepts a servo ID, and a value to write to the pin associated with that servo.

Servo IDs start at 0 and should be uniquely identified on every board, allowing a maximum of 255 servos on each bot, not including signals. They should be assigned different values because if commands are routed through a central board, it allows the protocol to remain the same. If an invalid servo ID is passed to any given

board, the board will still read another value from the serial port to avoid syncing issues, but it will do nothing to its servos.

Servo Values

The value passed to the servo can be from 0-180. If it exceeds 180 and is not a valid signal, the board will change the angle to 180. On positional rotation servos, this number represents the servo's target angle. On continuous rotation servos, 90 represents stillness, 180 represents full speed in one direction, and 0 represents full speed in the other.

Each board has callibrations for each servo that scale the angles from 0-180 to some minimum angle and some maximum angle. The host program has no need to know these values. When the board reports current values, they are non-scaled.

To ensure that the board is reading the correct information (servo ID or servo position), there is a cancel signal.

Signals

The boards accept some pre-defined signals that break the default flow. It can recieve these signals at any time and will terminate its current servo command. In addition, the board may print some signals to output. Both kinds of signals, incoming and outgoing, are assigned starting from 255 and going down. In the arduino code, incoming signals are denoted by ∗_**SIGNAL**, and the outgoing signals are denoted by ∗_**RESPONSE**. We will use this convention.

CANCEL_SIGNAL

This signal exists to cancel all pending input and syncronize the host and master. If a host connects to an Arduino and does not know its current state, it can send this signal and know that the board is waiting on a servo ID.

WAIT_RESPONSE

This response indicates that the board has no bytes left to read and is ready for input. A host does not need to wait for this response, but it may want to if it is experiencing difficulties or the board may have crashed.

START_RESPONSE, END_RESPONSE

These responses mark the beginning and end of a reply to some signal.

DUMP_SIGNAL

This signal makes the board return various information about itself and its servos. As of writing, it returns **S↩ TART_RESPONSE**, an ID, its number of servos, the IDs of each servo followed by the value last sent to that servo, and **END_RESPONSE**. If a serial connection has acceess to other boards and is able to send them commands, it may send more servo IDs and servo positions before **END_RESPONSE**. Board IDs start from 181 and go up. It is recommended that if we build more modular bots such as this one in the future using the same protocol, they recieve unique board identifiers. The response is likely to change in the near future as we add sensors and other types of devices. Be sure to regularly check this document.

1.5.1 Examples

```
1 CANCEL_SIGNAL 0 180 1 135 2 90
```

This would send the servo with ID 0 a value of 180, servo 1 135, and servo 2 90. The board would then respond with **WAIT_RESPONSE**.

```
1 CANCEL_SIGNAL 0 180 1 135 22 CANCEL_SIGNAL 4 45
```

This would send the servo with ID 0 a value of 180, servo 1 135, begin to read a command for servo 22 but cancel, then send 45 to servo 4.

```
1 CANCEL_SIGNAL 0 0 1 45 2 90 3 135 4 180 DUMP_SIGNAL
```

On a board with ID **BOARD_ID** and 5 servos identified as 0-4, this would return the response

```
1 START_RESPONSE BOARD_ID 4 0 0 1 45 2 90 3 135 4 180 END_RESPONSE WAIT_RESPONSE
```

```
1 CANCEL_SIGNAL 0 0 0 45 0 90 0 135 0 180
```

If this stream of bytes were sent instantly, it would essentially move servo 0 directly to 180 degrees (if it was a positional servo) because of the speed of the commands. However, if we inserted a slight delay, we could slowly move the servo from its starting position to its ending position.

```
1 CANCEL_SIGNAL 0 0 1 0 0 45 1 45 0 90 1 90 0 135 1 135 0 180 1 180
```

This would move servo 0 and servo 1 from their start to end positions. If we were to send these bytes immediately, they would essentially both move instantaneously to 180 degrees (if they were positional servos). However, if we were to insert a delay, they would both appear to move together slowly to their end location despite the delay between commands. Putting this functionality on the Arduino boards themselves would cause the boards to lock and use up resources, but formulating commands like this allows computation to occur on other systems.

1.5.2 Values

Board IDs

Right hand servo board	181
Right hand flex sensor board	182

Servo IDs

Right hand wrist	0
Right hand thumb	1
Right hand index finger	2
Right hand middle finger	3
Right hand ring finger	4
Right hand pinky finger	5

Signals

CANCEL_SIGNAL	255
WAIT_RESPONSE	254
DUMP_SIGNAL	253
START_RESPONSE	252
END_RESPONSE	251

See also

servo.ino rhand

Chapter 2

Namespace Documentation

2.1   rhand Namespace Reference

Documentation for the rhand.py script.

Functions

- def connect (port)

    Sets the global ser variable.
- def sweep (initial, servos, starts, ends, steps)

    Produces complex command chains.
- def unsweep (servos, ends, steps)

    Produces a command chain that resets all servos to 0.
- def servowrite (commands, delay)

    Writes commands to the serial port.
- def gesture (initial, servos, starts, ends, steps, delay)

    Sets the servos to a position, waits for input, then resets them to 0.
- def reset ()

    Immediately sets the servos to 0.
- def cmd ()

    Sends the input to the serial port until a blank line is given.
- def dump

    Retrieves information about the board.
- def peace

    Makes a peace sign.
- def ok

    Makes an ok sign.
- def grab

    Makes a fist.
- def rockon

    Makes a rock on sign.
- def wiggle

    Wiggles the fingers.
- def count ()

    Counts to 5 on the fingers.
- def demo ()

    Performs all movements.

Variables

- int **CANCEL_SIGNAL** = 255
- int **DUMP_SIGNAL** = 253
- int **START_RESPONSE** = 252
- int **END_RESPONSE** = 251
- int RHAND_ID = 181

    The identification byte of the right hand board.

- tuple ser = serial.Serial()

    The serial connection for input and output.

2.1.1   Detailed Description

Documentation for the rhand.py script.

An example script for controlling the right hand. This script requires python with a library called pyserial. After instally python 3.4, pyserial can be installed by calling

```
1 pip install pyserial
```

This may vary from system to system. See the `python` and `pyserial` websites for help.

To run the script, execute

```
1 python -i rhand.py
```

or in python

```
1 import rhand
```

This will allow you to call the functions from a command interpreter. It is recommended you experiment with all functions and view the source to understand how you might produce commands. Also try demo() while connected.

The movements included are quickly written and for demo purposes only.

See also

    servo.ino Values

2.1.2   Function Documentation

2.1.2.1   def rhand.cmd (   )

Sends the input to the serial port until a blank line is given.

Must enter integers between 0-255.

Definition at line 185 of file rhand.py.

2.1.2.2   def rhand.connect (*port*)

Sets the global ser variable.

Definition at line 54 of file rhand.py.

2.1.2.3 def rhand.count ( )

Counts to 5 on the fingers.

See also

> sweep()

Definition at line 275 of file rhand.py.

2.1.2.4 def rhand.demo ( )

Performs all movements.

Definition at line 283 of file rhand.py.

2.1.2.5 def rhand.dump (*servos =* 6)

Retrieves information about the board.

The command sends the DUMP_SIGNAL to the serial port and reads back the response. The response is described in servo.ino.

When you write your own similar function, timing is important.

We may need to increase the response delay on the Arduino if you are unable to recieve a response. View the source of this function to understand how it works, but keep in mind it is untested. Also keep in mind that the function is very non-general as it always reads the same amount of bytes instead of looking for the end signal.

Note

> While this function could be used to change dynamically between gestures instead of resetting the servos to 0, this function did not exist when this script was first written.

Definition at line 195 of file rhand.py.

2.1.2.6 def rhand.gesture (*initial, servos, starts, ends, steps, delay*)

Sets the servos to a position, waits for input, then resets them to 0.

See also

> sweep() servowrite()

Definition at line 168 of file rhand.py.

2.1.2.7 def rhand.grab (*delay =* 0.02)

Makes a fist.

See also

> gesture()

Definition at line 249 of file rhand.py.

2.1.2.8   def rhand.ok (*delay =* 0.01)

Makes an ok sign.

See also

> gesture()

Definition at line 245 of file rhand.py.

2.1.2.9   def rhand.peace (*delay =* 0)

Makes a peace sign.

See also

> gesture()

Definition at line 241 of file rhand.py.

2.1.2.10   def rhand.reset (   )

Immediately sets the servos to 0.

Definition at line 181 of file rhand.py.

2.1.2.11   def rhand.rockon (*delay =* 0.01)

Makes a rock on sign.

See also

> gesture()

Definition at line 253 of file rhand.py.

2.1.2.12   def rhand.servowrite (*commands,   delay*)

Writes commands to the serial port.

If **delay** is 0, immediately writes a list of ints to ser as bytes. Otherwise, writes a list of ints to ser as bytes one at a time.

Inserting a delay controls the speed of the movements.

See also

> connect()

Definition at line 151 of file rhand.py.

2.1.2.13   def rhand.sweep (*initial,   servos,   starts,   ends,   steps*)

Produces complex command chains.

The output returned from the function is formatted in such a way that the commands can be transmitted to the serial port at variable speeds.

The parameters must all be lists of the same length whose indexes correspond with **servos**.

Parameters

initial	An array of initial commands to append to.
servos	The servos to control.
starts	The current positions of the servos.
ends	The desired end positions.
steps	The values to use to decrement/increment. Converted to absolute value.

Returns

Upon success, returns the generated list of commands. The commands are surrounded by the CANCEL_↩
SIGNAL for syncing. The servos will always reach their exact final destination, unless if **steps** at the index of
the servo is 0. Upon error, raises **'Invalid sweep command'**. This indicates non-matching list lengths.

Example

```
1 sweep ([], [1,2], [180,0], [90,180], [90,45])
```
Returns

```
1 [CANCEL_SIGNAL, 1, 180, 2, 0, 1, 90, 2, 45, 2, 90, 2, 135, 2, 180, CANCEL_SIGNAL]
```

The protocol this follows is described in servo.ino. By intertwining commands in this way, a delay can be
inserted between bytes and fluid motion is still preserved.

Definition at line 59 of file rhand.py.

2.1.2.14 def rhand.unsweep (*servos, ends, steps*)

Produces a command chain that resets all servos to 0.

See also

> sweep()

Definition at line 143 of file rhand.py.

2.1.2.15 def rhand.wiggle (*n =* 90*, delay =* 0.01*, wiggles =* 1)

Wiggles the fingers.

See also

> sweep()

Definition at line 257 of file rhand.py.

2.1.3 Variable Documentation

2.1.3.1 rhand.ser = serial.Serial()

The serial connection for input and output.

Must be initialized.

See also

> connect()

Definition at line 52 of file rhand.py.

Chapter 3

File Documentation

3.1 arduino/sketch/servo/settings/rhand.h File Reference

Macros

- #define SERVOS 6
- #define BUFSIZE 32

Enumerations

- enum {
  VERBOSE = 0, BOARD_ID = 181, **DEBUG_RX_PIN** = 1, **DEBUG_TX_PIN** = 0,
  CMD_RX_PIN = 1, **CMD_TX_PIN** = 0, **CMD_SERIAL_BAUDRATE** = 9600, **DEBUG_SERIAL_BAUDRATE**
  = 9600 }

Functions

- int getServoIndexFromID (uint8_t servo_id)
- int getServoIDFromIndex (uint8_t servo_index)
- int getServoPinFromIndex (uint8_t servo_index)

Variables

- uint8_t limit [SERVOS][2]
- uint8_t default_pos [SERVOS] = { 0, 0, 0, 0, 0, 0 }
- uint8_t reverse [SERVOS] = { 1, 0, 0, 0, 0, 1 }
- uint8_t pin_offset = 2

3.1.1 Detailed Description

This is an example **settings.h** for servo.ino.

Every Arduino board on the bot is to be a simple servo controller that accepts commands from serial input, and so they are all to share the code located in servo.ino. Each board needs its own header like this one to define custom information about the board and each servo on the board. This is such a settings file and will be documented as an example.

3.1.2 Macro Definition Documentation

3.1.2.1 #define BUFSIZE 32

Size of the print buffer (must be large enough to hold DUMP_RESPONSE_LEN).

Definition at line 18 of file rhand.h.

3.1.2.2 #define SERVOS 6

Number of servos to assign.

Definition at line 14 of file rhand.h.

3.1.3 Enumeration Type Documentation

3.1.3.1 anonymous enum

Enumerator

> **VERBOSE**   Whether to print debug output to the serial port.
>
> **BOARD_ID**   The unique identification for this board.

Definition at line 54 of file rhand.h.

3.1.4 Function Documentation

3.1.4.1 int getServoIDFromIndex (uint8_t *servo_index*)

This function gets servo ID using its array index. If the servo index is out of bounds, it returns -1. This must correspond with getServoIndexFromID().

Definition at line 88 of file rhand.h.

3.1.4.2 int getServoIndexFromID (uint8_t *servo_id*)

This function gets array index of a servo using its ID. If the servo does not belong to this board, it returns -1. It is recommended that boards with complex ID assignments use a switch statement or a similar solution to save resources. This function must correspond with getServoIDFromIndex().

See also

> Values

Definition at line 75 of file rhand.h.

3.1.4.3 int getServoPinFromIndex (uint8_t *servo_index*)

This function returns the desired pin of a servo given its index number, allows complex pin assignment. If the servo idex is not valid, returns -1.

Definition at line 101 of file rhand.h.

3.1.5 Variable Documentation

3.1.5.1 uint8_t default_pos[SERVOS] = { 0, 0, 0, 0, 0, 0 }

The default position to set each servo to. Its index corresponds with servo.

Definition at line 42 of file rhand.h.

3.1.5.2 uint8_t limit[SERVOS][2]

Initial value:

```
= {
    { 15, 55 }, { 45, 120 }, { 35, 110 },
    { 15, 105 }, { 15, 120 }, { 35, 140 },
}
```

This is the callibration for each servo- its minimum then maximum angle (for positional servos) or its minimum then maximum speed (for continuous rotation servos). Its index corresponds with servo.

Definition at line 34 of file rhand.h.

3.1.5.3 uint8_t pin_offset = 2

Pin value to begin assigning pins at. Unique to this settings file.

See also

getServoPinFromIndex()

Definition at line 52 of file rhand.h.

3.1.5.4 uint8_t reverse[SERVOS] = { 1, 0, 0, 0, 0, 1 }

Indicates whether to reverse the angles for each servo, for servos that turn the wrong way. Its index corresponds with servo.

Definition at line 47 of file rhand.h.

3.2 arduino/sketch/servo/servo.ino File Reference

```
#include <Servo.h>
#include "settings.h"
#include <stdint.h>
```

Enumerations

- enum {
  MIN_LIM = 0, MAX_LIM = 1, CANCEL_SIGNAL = 255, WAIT_RESPONSE = 254,
  DUMP_SIGNAL = 253, START_RESPONSE = 252, END_RESPONSE = 251, DUMP_RESPONSE_LEN =
  SERVOS ∗ 2 + 4 }

    The print buffer.

Functions

- void setup ()
- void loop ()
- void setAdjustedAngles (uint8_t servo_index)
- uint8_t getAdjustedAngle (uint8_t servo_index, uint8_t servo_angle)
- void setServoFromIndex (uint8_t servo_index, uint8_t servo_angle)
- void setServoFromID (int servo_id, uint8_t servo_angle)
- void dump ()
- void serialWait ()

Variables

- Servo servo [SERVOS]
- uint8_t adjusted_angles [SERVOS][181]
- uint8_t current_pos [SERVOS] = { 0 }
- char **buf** [BUFSIZE]

3.2.1 Detailed Description

This is the common code for all Arduino servo boards.

The program is written to be a simple module that can pass values to servos. Each board needs its own settings file. Because of limitations of the Arduino command line interface, these are included as **settings.h**. The Makefile manages which configuration is currently located in the same directory as this file and named **settings.h**. A better build system in the future could use the tools **avrdude** or **ino**. Other Arduino boards in the project, such as flex sensor boards, use an identical folder structure and include process.

See rhand.h for an example of a settings header, and Arduino Protocol for information servo IDs and other relevant values.

3.2.2 Enumeration Type Documentation

3.2.2.1 anonymous enum

The print buffer.

Enumerator

> ***MIN_LIM***   Index for the limit variables.
>
> ***MAX_LIM***   Index for the limit variables.
>
> ***CANCEL_SIGNAL***   Incoming signal commanding termination of the current loop.
>
> ***WAIT_RESPONSE***   Outgoing signal indicating that we are waiting for input.
>
> ***DUMP_SIGNAL***   Incoming signal indicating that we are to print information to serial.
>
> ***START_RESPONSE***   Beginning of some multi-byte response.
>
> ***END_RESPONSE***   End of some response.
>
> ***DUMP_RESPONSE_LEN***   The length of the response to the dump signal.

Definition at line 45 of file servo.ino.

3.2.3 Function Documentation

3.2.3.1 void dump ( )

Writes various information about the board to the serial port. Triggered on reception of DUMP_RESPONSE_LEN. Transmits the ID of the board, the number of servos, each servo ID, and the current position of the servos.

See also

> Arduino Protocol, START_RESPONSE, END_RESPONSE, current_pos, getServoIDFromIndex()

Definition at line 251 of file servo.ino.

3.2.3.2 uint8_t getAdjustedAngle (uint8_t *servo_index,* uint8_t *servo_angle*)

Retrieves modified angles for a servo.

Returns

> The adjusted angle.

Parameters

servo_index	The servo index to use.
servo_angle	The angle to retrieve.

See also

> setAdjustedAngles(), adjusted_angles

Definition at line 178 of file servo.ino.

3.2.3.3 void loop ( )

The **loop** function is called continually until the program exits. It performs actions based on Arduino Protocol. Recieves two unsigned 8-bit integers, a servo id and a servo angle, then calls setServoFromID(). If the DUMP_SI←GNAL is recieved at any time, calls dump() and continues. If the CANCEL_SIGNAL is recieved, it does nothing and continues. If, at the beginning of the function, there is no pending input, it transmits WAIT_RESPONSE, then does nothing until input is available.

Definition at line 109 of file servo.ino.

3.2.3.4 void serialWait ( )

Waits for serial input to become available then returns.

Definition at line 277 of file servo.ino.

3.2.3.5 void setAdjustedAngles (uint8_t *servo_index*)

Calculates modified angles for a servo. The function scales angles from 0-180 to some minimum and maximum callibration, then stores them for later retrieval.

Parameters

servo_index	The servo to calculate angles for.

See also

> getAdjustedAngle(), adjusted_angles

Definition at line 154 of file servo.ino.

3.2.3.6 void setServoFromID (int *servo_id,* uint8_t *servo_angle*)

Writes an angle to a pin associated with a servo ID.

See also

> setServoFromIndex(), getServoIndexFromID()

Definition at line 230 of file servo.ino.

3.2.3.7 void setServoFromIndex (uint8_t *servo_index,* uint8_t *servo_angle*)

Writes an angle to a pin associated with a servo index. Servos are controlled by sending varying pulse widths over their signal wire. The adjusted angle will be sent to the servo. Updates current_pos with the non-adjusted angle. If the servo index is invalid, the function does nothing and exits, but if the angle is out of range, it is changed to a valid value.

Parameters

servo_index	The servo to write to.
servo_angle	The angle to write after adjusting. It may be reversed.

See also

> getAdjustedAngle(), reverse

Definition at line 195 of file servo.ino.

3.2.3.8 void setup ( )

The **setup** function is called at the beginning of the program. In it, we call setAdjustedAngles for each servo and assign a pin to each servo.

Definition at line 69 of file servo.ino.

3.2.4 Variable Documentation

3.2.4.1 uint8_t adjusted_angles[SERVOS][181]

The adjusted angles of the servos given the limit of each servo. A table calculated at initialization for optimization purposes. Its index corresponds to servo. The program uses the convention "angles", but continuous rotation servos are technically compatible.

See also

> setAdjustedAngles (), getAdjustedAngle ()

Definition at line 36 of file servo.ino.

3.2.4.2 uint8_t current_pos[SERVOS] = { 0 }

The current position of each servo. These are updated whenever a servos value is changed and retrieved upon DUMP_SIGNAL.

Definition at line 41 of file servo.ino.

3.2.4.3 Servo servo[SERVOS]

The representation of the servos. Used to keep track of pin assignments and send output.

See also

getServoIndexFromID(), Arduino

Definition at line 26 of file servo.ino.

Index