Team L205 IDP Documentation

Release 1.0

Eric Wieser, Matt Diesel

Dec 30, 2016

Contents

CHAPTER 1

Getting started

A guide for the electronics and mechanics team, when they need to test.

First of all, you'll want to get a copy of the code. We're using git for version control, so you can get that like this on the command line:

```
$ cd ~/idp_shared/<yourcrsid>
$ git clone ~/idp_shared/Common/repo.git
Initialized empty Git repository in /groups/IB/idp/idp-l205/efw27/repo/.git/
$ cd repo
```

After you've done this once, you should type the following every time the software team change the code:

```
$ cd ~/idp_shared/<yourcrsid>/repo
$ git pull
```

1.1 Useful programs

The following launches a keyboard interface for remote controlling the robot. Upon startup, it describes the key-mapping:

```
$ ./test t_remote
```

To test the competition code, use:

```
$ ./test t_all
```

Which allows you to enter the main routine at any point in the sequence

1.1.1 Calibration

To calibrate the eggsensor:

```
$ ./test dev/t_eggsensor_calib
constructed
initialized
Try and vary the ambient light while sampling
Place over brown egg, and hit enter
```

This program will expect you to place each egg until it in turn, and will take samples. Do this on the conveyor. Make sure to sample the edges of eggs as well as the centers. Also, spin the creme egg.

Running this will regenerate the `egg_stats.cc` file. To check the calibration, run the following, which will show which eggs are being read:

```
$ ./test t_eggidentify
none
none
brown
...
```

1.2 When things go wrong

You'll get errors if things aren't working. A common one is:

```
terminate called after throwing an instance of 'LinkError'
  what():  Host not found on network
./test: line 1: 22780 Aborted                 tests/$1.wifi
```

If this occurs, the robot is probably not yet powered. Wait for the blue LED. If all fails, pull the plug on it and try again.

Another one is:

```
terminate called after throwing an instance of 'PortError'
  what():  Port P1 disconnected.
./test: line 1: 22780 Aborted                 tests/$1.wifi
```

Which indicates a missing or broken electronics board. You can debug further with:

```
$ ./test t_conns
constructed
initialized
Testing P1... Connected
Testing P2... Connected
Testing P3... Disconnected
```

If all fails, then the controller itself probably needs its power taken away.

TL;DR: turn it off and on again

1.3 Building the docs

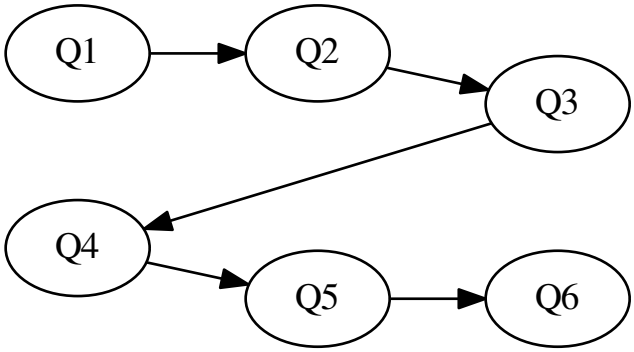
This documentation is autogenerated. Building it is slightly involved, and won't work on the department computers. It requires python and doxygen:

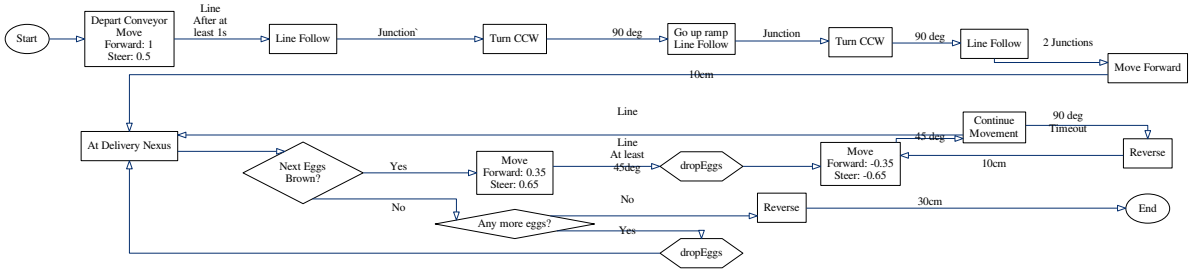
```
$ pip install breathe sphinx_rtd_theme   # first time only
$ cd docs
$ doxygen
$ make html  # or make.bat html on windows
```

The docs will be placed in `docs/_build/html`.

CHAPTER 2

Navigation plan





Functions

void **q1_collect_d2d3** (*Robot* &*r*)
[Q1] Collect D2/3 Eggs Start: Starting position End: Last position on conveyor

void **q2_deliver_d2d3** (*Robot* &*r*)
[Q2] Deliver D2/3 Eggs Start: Last position on conveyor End: Junction between D2/3 boxes

void **q3_return_from_d2d3** (*Robot* &*r*)
[Q3] Return from D2/3 Boxes Start: Junction between D2/3 boxes End: Starting square (at centre junction, facing west)

void **q4_collect_d1** (*Robot* &*r*)
[Q4] Collect D1 Eggs Start: Starting square (at centre junction, facing west) End: Last position on conveyor

void **q5_deliver_d1** (*Robot* &*r*)
[Q5] Deliver D1 Eggs Start: Last position on conveyor End: D1 box

void **q6_return_from_d1** (*Robot* &*r*)
[Q6] Return from d1 Start: D1 box End: Within starting area

2.1 Subroutines

Some bits of code are reused across multiple routes

Typedefs

typedef

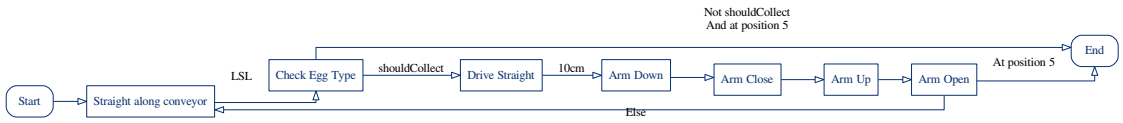
Functions

void **waitForLine** (*Robot* &*r*, *LineSensors*::*Reading*::State *s*)

void **waitForLine** (*Robot* &*r*, *negate* *n*)

void **goToConveyor** (*Robot* &*r*, bool *east* = true)

void **conveyorCollect** (*Robot* &*r*, *EGG_CALLBACK* *shouldCollect*)
Drives along the conveyor collecting eggs.



void **dropEggs** (*Robot* &*r*, int *n* = 1)
struct *#include* <*common.h*>**Public Members**

LineSensors::*Reading*::State `negate::s`

CHAPTER 3

Line following algorithm

void **followUntil** (*Robot* &*r*, float *distance*, linefollowTerminator **terminator* = & until_junction)

Follow a line until an event.

For example:

```
followUntil(robot, 0.6, until_junction)   // follow the line to a junction 60cm␣
↪away
followUntil(robot, 0.2, until_xjunction)  // follow the line to a cross-junction␣
↪20cm away
followUntil(robot, 0.2, until_bumper)     // follow the line until the bumper is␣
↪pressed 20cm away
followUntil(robot, 0.2, NULL)             // follow the line for exactly 20cm
```

Parameters

- `r`: A reference to a *Robot*
- `distance`: The expected distance to drive, in meters, before the event occurs
- `terminator`: A function taking `(Robot&, const LineSensors::Reading&)` that returns true when the robot should stop. If NULL, stop after `distance` has been travelled

Exceptions

- `LineLost`: The target line couldn't be found, recovery failed.
- *`Timeout::Expired`*: It's taking more than 25% longer than expected to reach the target

void **turnAtJunction** (*Robot* &*r*, int *turns*, bool *goForward* = true)

Turns the robot at a junction.

Parameters

- `r`: A reference to a *Robot*
- `turns`: The number of 90 degree turns to go through, counter-clockwise being positive.

- `goForward`: `false` if the robot already has its wheelbase over the junction

Exceptions

- `LineLost`: The target line couldn't be found, recovery failed.

CHAPTER 4

Color classification

The egg sensor reads four variables describing the egg. A range of similar eggs can be sampled to calibrate the sensor. From these readings, a normal distribution can be fitted to each variable and egg. Considering covariances, we can then generalize to a 4-variable normal distribution for each egg.

To identify the egg, the find the normal distribution with the highest probability density at a given point in variable-space.

The following steppable animation shows how this works for just two variables (red and blue).

The conversion of the raw recorded data to a set of normal distribution parameters is done with a small python script, that leverages the *numpy* numeric toolkit, and generates the *egg_stats.cc* file.

4.1 Api documentation

enum type `EggType`
Values:

std::array<*MultivariateNormal*<4>, *EGG_TYPE_COUNT*> `egg_stats::`**`expectations`**
We model each egg as a normal distribution over all readings for that egg.

The four variables of the distribution are the red, blue, white, and ambient components of the reading

The parameters for these models are in egg_stats.cc, which is auto-generated by a python script from a set of calibration readings

template <int *N*>
struct Represents a generalized normal distribution over N variables, described by the mean and covariance matrices.

Public Functions

double `MultivariateNormal::`**`mahalanobisDistanceSq`**(Matrix<float, N, 1> *value*) **const**
generalization of $\frac{x-\mu}{\sigma}^2$ to N variables

see: http://en.wikipedia.org/wiki/Mahalanobis_distance

Public Members

Matrix<float, N, 1> `MultivariateNormal::`**`mean`**

Matrix<float, N, N> `MultivariateNormal::`**`covariance`**

CHAPTER 5

Hardware access layer

To ensure hardware is operated correctly, each independant electronic subsystem is encapsulated in a class. This also allows us to make meaningful constructors and destructors, such as: setting up ports for inputs; turning off LEDs at shutdown; driving a motor slowly to hold the deliverer at startup.

These all get wrapped in a single *Robot* instance, with members as follows.

struct **Public Functions**

`Robot::Robot` (*RLink* &*rlink*)

Public Members

Drive `Robot::drive`

Arm `Robot::arm`

LineSensors `Robot::ls`

EggSensor `Robot::detector`

Courier `Robot::courier`

Bumper `Robot::bumper`

5.1 Actuators

Things which cause parts of the robot to move

class Interface to the two-wheeled drive system. Inherits from *Device* **Public Functions**

`Drive::Drive` (*RLink* &*r*, *Configuration* *c* = Drive::_defConfig)

Initialize a drive over a connection.

Parameters

- `r`: the link to the robot
- `c`: the drive geometry and speeds, used to populate *Drive::maxSpeeds*

void `Drive::move`(move_args *args*)

Should ensure that abs(args.forward) + abs(args.steer) <= 1

Parameters

- `args.forward`: non-dimensional linear speed: 1 is full speed forwards, -1 is full speed backwards
- `args.steer`: non-dimensional rotational speed: 1 is full speed CCW

Timeout `Drive::straight`(float *dist*, float *speed* = 1)
Move in a straight line, and return a timeout indicating expected completion.

Timeout `Drive::turn`(float *angle*, float *speed* = 1)
Turn an angle on the spot, and return a timeout indicating expected completion.

void `Drive::setWheelSpeeds`(float *left*, float *right*)
low-level motor access. *Speeds* should be between 1 and -1

void `Drive::stop`()
shorthand for no motion

Public Members

Speeds `Drive::maxSpeeds`
The maximum speeds the robot is able to acheive.

Public Static Functions

uint8_t `Drive::convertSpeed`(float *s*)
convert floating point speed to sign/magnitude

struct Describes the physical configuration of the robot. **Public Members**

float `Drive::Configuration::radius`
wheel radius, in m

float `Drive::Configuration::spacing`
distance between centers of wheels, in m

float `Drive::Configuration::rpm`
motor speed, in rpm

struct struct indicating maximum speeds, built from a *Configuration*

class Interface to the egg-grabbing arm of the robot. Inherits from *Device* **Public Functions**

`Arm::Arm`(*RLink* &*r*, port::*Name* *name*)

void `Arm::up`()

void Arm::**down**()

void Arm::**open**()

void Arm::**close**()
class Interface to the runner holding the eggs, its indicator LEDs, the light gate that verifies the presence of an egg, and the bucket at the end of the runner which delivers the eggs into their cups. Affectionately known as the courier, as it carries things Inherits from *Device* **Public Functions**

void Courier::**recordEggAdded**(*EggType* *e*)
Indicate that a new egg has been added to the rail.

This updates the internal record of currently-held eggs, and turns on the appropriate LEDs.

void Courier::**unloadEgg**()
Unload the egg at the bottom of the stack, updating state and LEDs.

EggType Courier::**egg**(int *n*) **const**
type of the egg n from the bottom

int Courier::**volume**() **const**
The number of eggs on the rail.

bool Courier::**eggDetected**() **const**
if an egg is at the bottom of the courier

5.2 Sensors

Things which give the robot information about its surroundings
class Interface to the LEDs and LDR comprising the egg sensor. Includes the algorithm for identifying eggs Inherits from *Device* **Public Functions**

EggSensor::**EggSensor**(*RLink* &*r*, port::*Name* *port*)

EggSensor::*Reading* EggSensor::**read**(int *samples* = 5)
read the sensor, taking an average over multiple samples
struct **Public Members**

uint8_t EggSensor::Reading::**r**
reflection from red LED

uint8_t EggSensor::Reading::**b**
reflection from blue LED

uint8_t EggSensor::Reading::**w**
reflection from white LED

uint8_t EggSensor::Reading::**a**
ambient reading

std::array<float, *EGG_TYPE_COUNT*> EggSensor::Reading::**probabilities**
"distances" to each egg. Lower values indicate greater likelihood

EggType `EggSensor::Reading::`**`bestGuess`**
shorthand for most likely egg type

class Interface to the three front-mounted line sensors. Inherits from *Device* **Public Functions**

`LineSensors::`**`LineSensors`**(*RLink* &*r*, port::*Name* *p*)

LineSensors::*Reading* `LineSensors::`**`read`**()

struct **Public Members**

bool `LineSensors::Reading::`**`lsl`**
left sensor reading

bool `LineSensors::Reading::`**`lsc`**
right sensor reading

bool `LineSensors::Reading::`**`lsr`**
center sensor reading

bool `LineSensors::Reading::`**`lsa`**
arm sensor reading

float `LineSensors::Reading::`**`position`**
Line position, where between -1 and 1, with left positive.

+-Inf and NaN indicate a lost line

class Interface to the limit switch bumper on the front of the robot. Inherits from *Device* **Public Functions**

`Bumper::`**`Bumper`**(*RLink* &*r*, port::*Name* *port*)

Bumper::*Reading* `Bumper::`**`read`**()

struct **Public Members**

bool `Bumper::Reading::`**`left`**
left switch is pressed

bool `Bumper::Reading::`**`right`**
right switch is pressed

float `Bumper::Reading::`**`position`**
1 for left, -1 for right, 0 for straight, and NaN for not pressed

For ease of debugging, some of these readings have `ostream <<` overloads, to allow:

```
std::cout << robot.ls.read() << std::endl
```

5.3 Low level

All of the above classes use the following utility classes to interface with the hardware.

class Wraps robot_link to indicate failures by throwing a *LinkError* object. Inherits from robot_link **Public Functions**

void `RLink::`**`initialise`**()
Initialise the link by the most appropriate method for the location the code is running.

void RLink::**command**(command_instruction *cmd*, int *arg*)
Send a command to the robot.

int RLink::**request**(request_instruction *req*)
Request data from the robot.

uint8_t RLink::**status**()
Get the status register, a bitfield containing {comm_err, i2c_err, es_trig, es_mode, moving, ramped, _, _}

Does not throw *LinkError*

class Base class for all devices which require a link to the robot. Subclassed by *Arm*, *Bumper*, *Courier*, *Drive*, *EggSensor*, *LineSensors*, *Port* **Protected Attributes**

RLink &Device::**_r**
internal reference to a robot connection

type port::**Name**
An enum of port names, from P0 to P7, and PA0 to PA7

class Interface to a set on pins on a particular port. Allows masking of pins, to allow multiple *Device*s to share a I2C port without interfering with each other's bits Provides operator overloading for simple use:

```
Port sensor(rlink, port::P2, 0xF);  // bottom 4 bits of port 2
uint8_t reading = sensor;  // read sensor
sensor = 0x42;  // write to sensor
```

Note that conversion to an int will return the current input, which is not necesarily the previous output

Inherits from *Device*

Public Functions

Port::**Port**(*RLink* &*r*, port::*Name* *p*, uint8_t *mask* = 0xFF)
Create a port over the connection r, using the port with address p.

Optionally specify a set of bits mask to restrict the scope of this instance to. Throws *PinsDoublyMapped* if multiple instances attempt to use the same ports

Port::**operator uint8_t**() **const**
Read a word to the port, keeping only the bits specified in the mask.

void Port::**operator=**(uint8_t *val*)
Write a word to the port, touching only the bits specified in the mask.

5.3.1 Exceptions

To prevent errors silently occuring without being noticed (or worse, error codes being handled as values), exceptions are used for all critical errors. These all derive from *std::exception*, and implement the *const char* what()* member to give a brief summary of the error to the programmer, to allow them to fix the appropriate electrical/network problem.

class Thrown when an *RLink* command or request goes wrong. Contains the original error code Inherits from exception Subclassed by *PortError* **Public Functions**

virtual const char *LinkError::**what**() **const**
override of std::exception::what()

Public Members

const link_err `LinkError::`**`err`**
The original error code.

const bool `LinkError::`**`is_fatal`**
Whether the error is marked as fatal by robot_link.

const bool `LinkError::`**`is_i2c`**
If true, indicates that the error has no code, and is instead a bus error.

struct Specialization of *LinkError*, thrown when an I2C error occurs when accessing a port. Typically implies loss of electrical connection Inherits from *LinkError* **Public Functions**

virtual const char *`PortError::`**`what`**`()` **const**
override of std::exception::what()

Public Members

const *port*::Name `PortError::`**`port`**
the disconnected port

struct Specialization of *LinkError*, thrown when an I2C error occurs when accessing a port. Typically implies loss of electrical connection Inherits from exception **Public Members**

const *port*::Name `PinsDoublyMapped::`**`port`**
the port causing the issue

const uint8_t `PinsDoublyMapped::`**`pins`**
the mask of pins that have already been allocated

CHAPTER 6

Utilities

class Class for keeping track of expected times for operations. Example usage:

```
using namespace std::literals::chrono_literals;

Timeout timeout(2s);
try {
    doAThing();
    timeout.check();
    do {
        keepGoing();
        timeout.check();
    } while (stillGoing())
} catch(Timeout::Expired) {
    std::cout << "took too long" << std::endl;
```

Public Functions

`Timeout::Timeout`(duration_type *duration*)
: create a timeout `duration` in the future

`Timeout::Timeout`(clock::time_point *end*)
: create a timeout ending at `end`

void `Timeout::check`() **const**
: check if the timeout has expired, and throw *Expired* if so

void `Timeout::wait`() **const**
: wait for the timeout to expire

class Inherits from exception

class Heirarchical logger, use to produce indented logs. **Public Functions**

`Logger::~Logger`()
: upon destruction, log either "[done]" or "[threw]"

Logger Logger::**child**(std::string *name*)
create a sublogger of this logger

void Logger::**checkpoint**(*Robot* &*r*, std::string *id*)
record a checkpoint. Scope for stopping the robot and waiting for user interaction

int Logger::**depth**() **const**
the depth of this logger - used for indentation

Public Static Functions

static *Logger* &Logger::**active**()
get the current active logger

Friends

template <typename T>
std::ostream &**operator<<**(*Logger* &*logger*, **const** T &*t*)
Output content to the logger, prefixed with appropriate indentation.
class

- genindex

Index

M

N

O

P

Q

R

T

W