MAGL Micromouse

Generated by Doxygen 1.8.13

Contents

Chapter 1

Micromouse

This software was deveoped by Nicholas Appleton and Christian Woof of UWE Robotics. It is for use with a micromouse with a dsPIC30F4011 microcontroller, with the pin-layout as shown in Fig. 1. It is desined with compliance to the micromouse international competition. The maze that it was created to solve is an 8x6 maze, however, the size of the maze can be set accordingly and the actual size of the maze does not need to be known, as long as the size is set to greater than the actual size. A simulator is also included in the code so that the functionality can be tested without a physical mouse. To use the simulator, the integration folder should be excluded from the project, and the SIMULATOR macro set to 1. A series of test mazes can be found later in this document, these should be pasted into the simulator.c file.

The Maze contains a high level algorithm that maps the maze by pushing any cell it finds to an openlist of unexplored cells, from which the most recently found is explored. A nodemap is created where Nodes are created and destroyed as necissary to the final solving. Various functionality is shown in Fig. 2.

Fig. 3 shows the interaction between the high level algorithm and the low level integration code.

Test Mazes

Test mazes for use in the simulated environment. Each maze focusses on tripping up the algorithm in a different way.

1) {0,1,1,1, 0,1,1,1, 0,1,1,1, 0,1,1,1, 0,1,1,1, 0,1,1,1}, {0,0,0,1, 0,1,0,0, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 0,0,0,1, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,1,0,0}, {0,1,0,1, 1,1,0,1, 0,0,1,1, 0,1,0,0, 0,1,1,1, 0,1,0,1}, {0,1,0,1, 0,1,1,1, 1,0,0,1, 1,1,0,0, 0,0,0,1, 0,1,0,0}, {0,1,0,1, 1,0,0,1, 1,0,1,0, 1,0,1,0, 1,1,0,0, 0,1,0,1}, {0,0,0,1, 1,0,1,0, 1,0,1,0, 0,0,1,0, 1,0,1,0, 0,1,0,0}, {1,0,0,1, 1,0,1,0, 1,1,1,0, 1,1,0,1, 1,0,1,1, 1,1,0,0}

2) {0,1,1,1, 0,0,1,1, 0,0,1,0, 0,0,1,0, 0,1,1,0, 0,1,1,1}, {0,0,0,1, 0,1,0,0, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 0,0,0,1, 1,0,0,1, 1,0,0,0, 0,1,0,0, 0,1,0,1}, {0,0,0,1, 0,1,0,0, 0,0,1,1, 0,1,1,0, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 0,1,0,1, 1,0,0,1, 1,0,0,0, 0,1,0,0, 0,1,0,1}, {0,0,0,1, 0,0,0,0, 1,0,1,0, 0,0,1,0, 1,0,0,0, 0,1,0,0}, {0,1,0,1, 1,0,0,1, 0,1,1,0, 0,0,0,1, 0,0,1,0, 0,1,0,0}, {1,0,1,1, 1,0,1,0, 1,0,0,0, 1,1,0,0, 1,1,0,1, 1,1,0,1}

3) {0,1,1,1, 0,0,1,1, 0,1,1,0, 0,0,1,1, 1,0,1,0, 0,1,1,0}, {0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,0,1,1, 1,1,0,0}, {0,1,0,1, 0,1,0,1, 1,0,0,1, 1,1,0,0, 1,0,0,1, 0,1,1,0}, {0,1,0,1, 0,1,0,1, 0,0,1,1, 0,0,1,0, 0,1,1,0, 0,1,0,1}, {0,1,0,1, 0,1,0,1, 1,0,0,1, 1,1,0,0, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 0,1,0,1, 0,0,1,1, 0,1,1,0, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 1,0,0,1, 1,1,0,0, 1,0,0,1, 1,1,0,0, 0,1,0,1}, {1,0,0,1, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,1,0,0}

4) {0,1,1,1, 0,0,1,1, 0,0,1,0, 1,0,1,0, 0,1,1,0, 0,1,1,1}, {1,0,0,1, 0,1,0,0, 0,1,0,1, 0,0,1,1, 1,1,0,0, 0,1,0,1}, {0,0,1,1, 1,0,1,0, 1,0,0,0, 1,0,0,0, 0,0,1,0, 1,1,0,0}, {0,1,0,1, 0,0,1,1, 0,0,1,0, 0,1,1,0, 0,1,0,1, 0,1,1,1}, {0,0,0,1, 1,1,0,0,

1,0,0,1, 1,1,0,0, 0,1,0,1, 0,1,0,1}, {1,0,0,1, 0,1,1,0, 0,0,1,1, 0,0,1,0, 0,0,0,0, 0,1,0,0}, {0,1,1,1, 1,0,0,1, 1,1,0,0, 0,1,0,1, 0,1,0,1, 0,1,0,1}, {1,0,0,1, 1,0,1,0, 1,0,1,0, 1,1,0,0, 1,1,0,1, 1,1,0,1}

5) {0,1,1,1, 0,1,1,1, 0,1,1,1, 0,0,1,1, 1,0,1,0, 0,1,1,0}, {0,0,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,1,0, 0,1,0,0}, {0,1,0,1, 0,1,0,1, 1,1,0,1, 1,0,0,1, 0,1,1,0, 0,1,0,1}, {0,1,0,1, 0,1,0,1, 0,0,1,1, 0,1,1,0, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 0,1,0,1, 0,0,0,1, 1,1,0,0, 0,1,0,1, 0,1,0,1}, {0,1,0,1, 1,0,0,1, 0,1,0,0, 0,0,1,1, 1,0,0,0, 0,1,0,0}, {0,0,0,1, 1,1,1,0, 1,0,0,1, 0,0,0,0, 1,0,1,0, 0,1,0,0}, {1,1,0,1, 1,0,1,1, 1,0,1,0, 1,0,0,0, 1,0,1,0, 1,1,0,0}

6) {0,1,1,1, 0,0,1,1, 1,1,1,0, 0,0,1,1, 1,1,1,0, 0,1,1,1}, {0,0,0,1, 0,0,0,0, 1,0,1,0, 0,1,0,0, 0,1,1,1, 0,1,0,1}, {0,1,0,1, 0,0,0,1, 1,1,1,0, 1,0,0,1, 0,0,0,0, 0,1,0,0}, {1,1,0,1, 0,1,0,1, 0,0,1,1, 0,1,1,0, 0,1,0,1, 0,1,0,1}, {0,0,1,1, 1,1,0,0, 0,0,0,1, 1,1,0,0, 1,1,0,1, 0,1,0,1}, {0,1,0,1, 0,1,1,1, 0,1,0,1, 0,0,1,1, 1,0,1,0, 0,1,0,0}, {1,0,0,1, 0,0,0,0, 0,0,0,0, 1,0,0,0, 1,0,1,0, 0,1,0,0}, {1,0,1,1, 1,1,0,1, 1,0,0,1, 1,0,1,0, 1,0,1,0, 1,1,0,0}

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

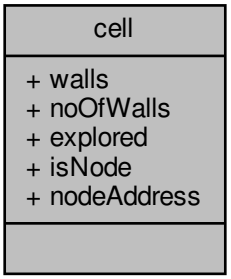
Class Documentation

4.1 cell Struct Reference

All info about a given cell.

`#include <MappingFunctions.h>`

Collaboration diagram for cell:



Public Attributes

- unsigned int walls: 4
- unsigned int noOfWalls: 6
- unsigned int explored: 1
- unsigned int isNode: 1
- unsigned char nodeAddress

4.1.1 Detailed Description

All info about a given cell.

4.1.2 Member Data Documentation

4.1.2.1 explored

```
unsigned int cell::explored
```

Marks whether cell has been visited

4.1.2.2 isNode

```
unsigned int cell::isNode
```

Marks whether cell is a Node or not

4.1.2.3 nodeAddress

```
unsigned char cell::nodeAddress
```

Index of Node that references this cell in nodelist

4.1.2.4 noOfWalls

```
unsigned int cell::noOfWalls
```

Number of walls of cell. can never be more than 4

4.1.2.5 walls

```
unsigned int cell::walls
```

Layout of walls; 1 denotes a wall, 0 is no wall. bit-order is N>E>S>W

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

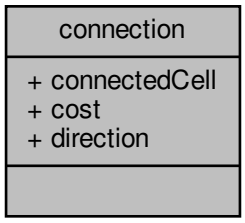
- Algorithm/MappingFunctions.h

4.2 connection Struct Reference

Connection between 2 nodes.

`#include <MappingFunctions.h>`

Collaboration diagram for connection:



Public Attributes

- unsigned char connectedCell
- unsigned int cost
- unsigned int direction: 4

4.2.1 Detailed Description

Connection between 2 nodes.

4.2.2 Member Data Documentation

4.2.2.1 connectedCell

`unsigned char connection::connectedCell`

Index of the connected Node in th maze

4.2.2.2 cost

`unsigned int connection::cost`

Cost to get to connected Node

4.2.2.3 direction

```
unsigned int connection::direction
```

direction to exit cell to get to connected Node

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

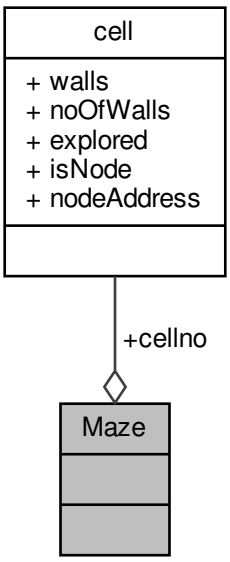
- Algorithm/MappingFunctions.h

4.3 Maze Struct Reference

Contains the representation of the maze itself.

```
#include <MappingFunctions.h>
```

Collaboration diagram for Maze:



Public Attributes

- cell cellno [HEIGHT][WIDTH]

4.3.1 Detailed Description

Contains the representation of the maze itself.

4.3.2 Member Data Documentation

4.3.2.1 cellno

```
cell Maze::cellno[HEIGHT][WIDTH]
```

Array of cells used as the maze representation

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

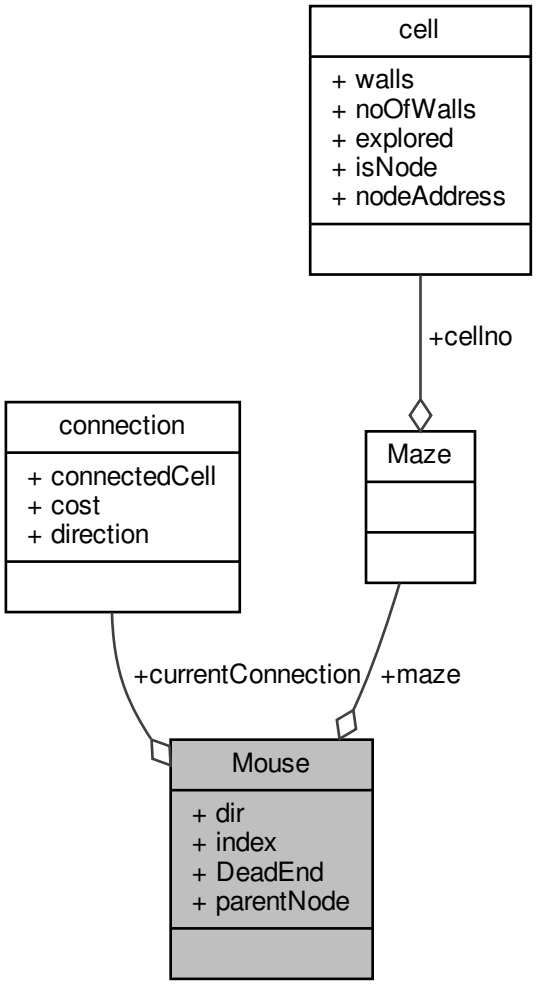
- Algorithm/MappingFunctions.h

4.4 Mouse Struct Reference

Representation of the Mouse in virtual space.

```
#include <MapMaze.h>
```

Collaboration diagram for Mouse:



Public Attributes

- unsigned int dir: 4
- unsigned char index
- unsigned int DeadEnd: 1
- struct Maze * maze
- unsigned char parentNode
- struct connection currentConnection

4.4.1 Detailed Description

Representation of the Mouse in virtual space.

represents the mouse that inhabits the virtual maze. including physical attributes and debugging info.

4.4.2 Member Data Documentation

4.4.2.1 currentConnection

```
struct connection Mouse::currentConnection
```

Info about current exploration from parent Node

4.4.2.2 DeadEnd

```
unsigned int Mouse::DeadEnd
```

Marks whether backtracking from a dead end

4.4.2.3 dir

```
unsigned int Mouse::dir
```

Direction the mouse is facing

4.4.2.4 index

```
unsigned char Mouse::index
```

Position of the mouse within the maze

4.4.2.5 maze

```
struct Maze* Mouse::maze
```

contains the mouse's model of the maze

4.4.2.6 parentNode

```
unsigned char Mouse::parentNode
```

index of Node last viseted in nodelist, next node found will be connected to this

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

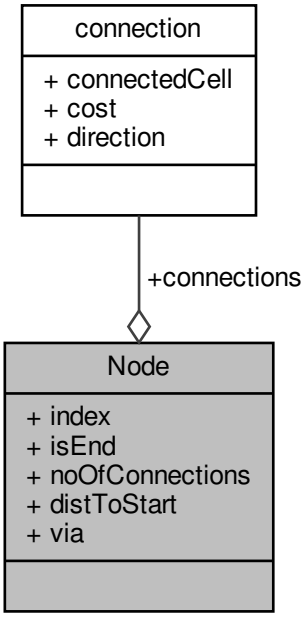
- Algorithm/MapMaze.h

4.5 Node Struct Reference

All info about a given Node.

`#include <MappingFunctions.h>`

Collaboration diagram for Node:



Public Attributes

- unsigned char index
- unsigned int isEnd: 1
- unsigned int noOfConnections: 3
- struct connection connections [4]
- int distToStart
- unsigned char via

4.5.1 Detailed Description

All info about a given Node.

4.5.2 Member Data Documentation

4.5.2.1 connections

```
struct connection Node::connections[4]
```

Array of connected nodes

4.5.2.2 distToStart

```
int Node::distToStart
```

Shortest distance found to centre of maze. -1 represents infinite distance

4.5.2.3 index

```
unsigned char Node::index
```

index of the cell that this node references

4.5.2.4 isEnd

```
unsigned int Node::isEnd
```

Marks whether the Node is at the centre of the maze

4.5.2.5 noOfConnections

```
unsigned int Node::noOfConnections
```

Number of Nodes connected to this one

4.5.2.6 via

```
unsigned char Node::via
```

Node shortest route goes through to get here.

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

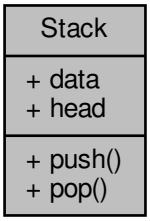
- Algorithm/MappingFunctions.h

4.6 Stack Struct Reference

array of data that is the Stack.

`#include <Stacks.h>`

Collaboration diagram for Stack:



Public Member Functions

- void push (Stack *stack, unsigned char Newdata)

  pushes item onto a stack.
- unsigned char pop (Stack *stack)

  pops an item from the top of the stack.

Public Attributes

- unsigned char data [WIDTH *HEIGHT *2]
- unsigned char head

4.6.1 Detailed Description

array of data that is the Stack.

The size of the stack equal to number of cells in the maze.

4.6.2 Member Function Documentation

4.6.2.1 pop()

```
unsigned char pop (
            Stack * stack )
```

pops an item from the top of the stack.

returns the data from the top-most stackitem, points the stack pointer to the item below where it was and frees the topmost item.

Parameters

stack	the stack from which the data will be popped.

4.6.2.2 push()

```
void push (
              Stack * stack,
              unsigned char Newdata )
```

pushes item onto a stack.

creates a new stackitem which contains the new data and points to the current head of the stack. The stack pointer is then moved to point at the new stackitem.

Parameters

stack	the stack that the data will be pushed to.
Newdata	the data that will be pushed to the stack.

4.6.3 Member Data Documentation

4.6.3.1 data

```
unsigned char Stack::data[WIDTH *HEIGHT *2]
```

data that is stored in the Stack

4.6.3.2 head

```
unsigned char Stack::head
```

head of the Stack where data is pushed to and popped from

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

- Stacks.h

Chapter 5

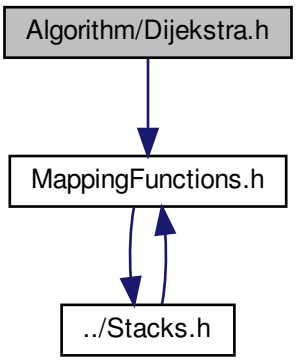
File Documentation

5.1 Algorithm/Dijekstra.h File Reference

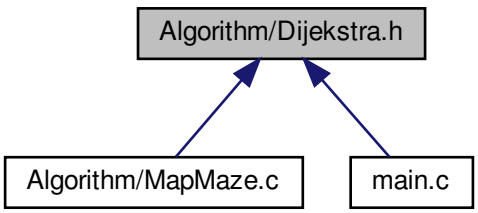
dijekstra's function and cocktail sort.

```
#include "MappingFunctions.h"
```

Include dependency graph for Dijekstra.h:



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



Functions

- Stack dijekstra (struct Maze *maze, Node nodemap[MAX_NODES], Node *start, Node *end, char startdir)

  Find fastest route between 2 Nodes.
- void cocktail (Node *arr[MAX_NODES])

  cocktail sorts given array.

5.1.1 Detailed Description

dijekstra's function and cocktail sort.

Cocktail sort is used to order the priority queue after each time a Node has finished being checked by the dijekstra's function.

Author

Nick Appleton @ UWE Robotics

Date

16/3/19

5.1.2 Function Documentation

5.1.2.1 cocktail()

```
void cocktail (
            Node * arr[MAX_NODES] )
```

cocktail sorts given array.

Parameters

arr	array to be sorted

5.1.2.2 dijekstra()

```
Stack dijekstra (
            struct Maze * maze,
            Node nodemap[MAX_NODES],
            Node * start,
            Node * end,
            char startdir )
```

Find fastest route between 2 Nodes.

Uses dijekstra's algorithm to find the shortest route thorugh the Nodemap. It routes through the maze, pushing each move that needs to be made to a stack. This stack is then returned.

The stack's format is: initial on-the-spot turn to face the correct direction, then it has the initial forward move (or 0 if there isn't one). From there it alternates between each forward move and each turn required to get into the next cell. The turns include the move into the Next cell. Last item in stack is a straight, moves into the end Node cell.

Parameters

maze	maze to search through
nodemap	nodemap of the maze given
start	location of the Node to start from
end	location of the Node to get to
startdir	direction the mouse is facing at the start

Returns

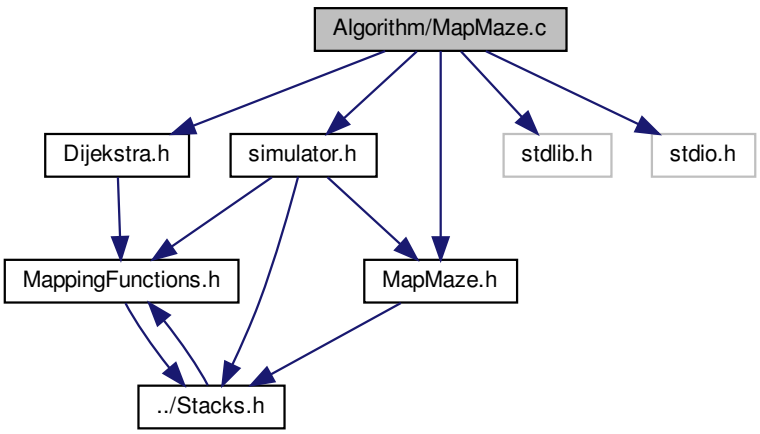
satck of moves to get from start to finish

5.2 Algorithm/MapMaze.c File Reference

Fully maps the maze.

```
#include "simulator.h"
#include "MapMaze.h"
#include "Dijekstra.h"
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for MapMaze.c:



Macros

- #define SIMULATOR 1

  Use simulator or Mouse.

Functions

- void mapmaze (struct Maze *mazeArg, Node *nodelist)

  main maze-mapping function.
- void SetupMapping (Stack *openlist, Node *nodelist)

  sets up the mouse ready to map out the maze.
- unsigned char createNode (unsigned char index, Node *nodelist)

  creates a new node.
- void checkcurrentcell (Stack *openlist, Node *nodelist, Stack *history)

  updates info on the cell currently occupied.
- void ConnectNodes (Node *nodelist, unsigned char dir)

  Connects the parent node to the current cell.
- void ExploreNewCell (Stack *openlist, Stack *history, Node *nodelist)

  Used to get to new areas.
- unsigned char identifyDirection (unsigned char target)

  identify in which direction a cell is.
- void moveToAdjacentCell (unsigned char direction)

  move mouse into an adjacent cell in the direction given.
- void virtualMouse (Node *nodelist)

  corrects any known but unexplored dead-ends.
- void VMcheck (unsigned char index, Node *nodelist)

  checks one cell for dead end and corrects it.
- void DestroyNode (Node *nodelist, unsigned char index)

  destroy a given Node.

Variables

- Mouse mouse

  global representation of the mouse used by every function.

5.2.1 Detailed Description

Fully maps the maze.

Maps the full maze including placing nodes and finding the connections between those Nodes.

Author

Nick Appleton @ UWE Robotics

Date

24/2/19

5.2.2 Macro Definition Documentation

5.2.2.1 SIMULATOR

```
#define SIMULATOR 1
```

Use simulator or Mouse.

If the variable is set to 1, the simulator will be used. Otherwise the target will be the actual mouse.

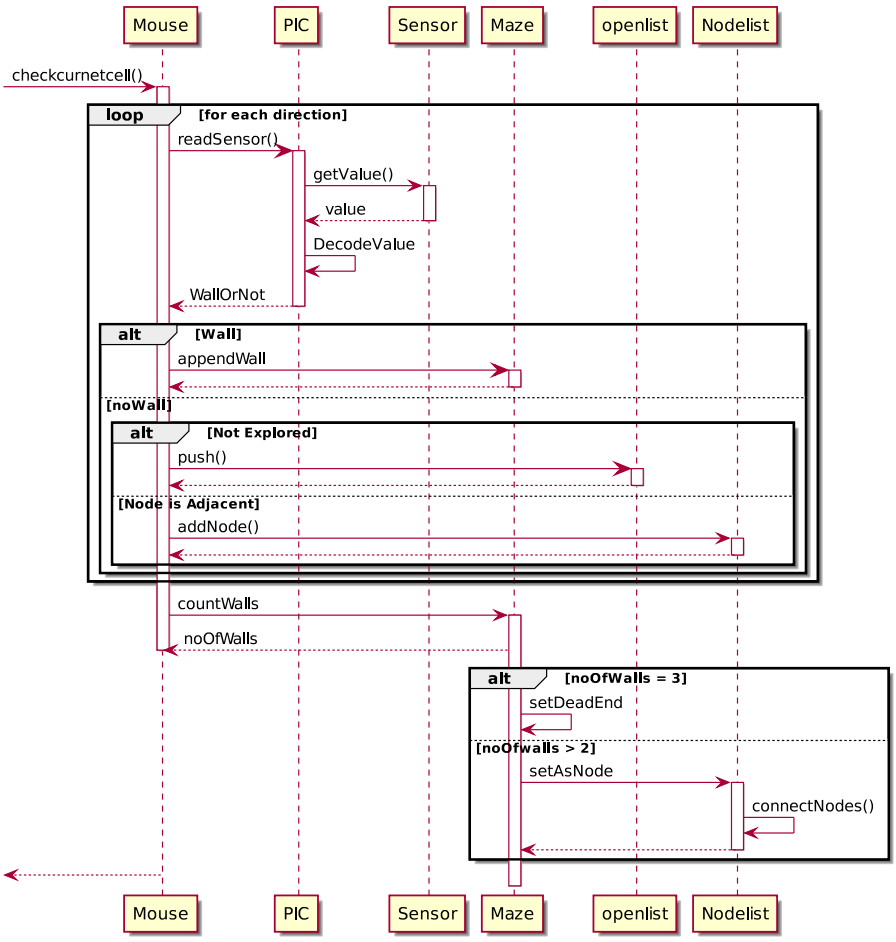
5.2.3 Function Documentation

5.2.3.1 checkcurrentcell()

```
void checkcurrentcell (
            Stack * openlist,
            Node * nodelist,
            Stack * history )
```

updates info on the cell currently occupied.

This includes correcting all the walls in the current cell and all adjacent cells, adding unexplored adjacent cells to the openlist and changing the cells properties to reflect it's status (it's now been explored).



Parameters

openlist	The stack of cells to be explored.
nodelist	list of all the Nodes in the maze.
history	stack of the cells which were visited by the mouse.

5.2.3.2 ConnectNodes()

```
void ConnectNodes (
            Node * nodelist,
            unsigned char dir )
```

Connects the parent node to the current cell.

Adds a connection to the parent Node to the current cell, also adds the connection back from current cell to parent Node. If the current cell is not a Node, it creates a new node to use.

Parameters

nodelist	List of all the Nodes in the maze.
dir	direction in which the mouse entered the cell to be connected to the parent.

5.2.3.3 createNode()

```
unsigned char createNode (
            unsigned char index,
            Node * nodelist )
```

creates a new node.

Initialises all the Node variables, sets the cell at the correct index to a node and adds a pointer to the new node.

Parameters

index	index at which the new Node is to be created.
nodelist	List of all the Nodes in the maze.

Returns

a pointer to the newly created node.

5.2.3.4 DestroyNode()

```
void DestroyNode (
            Node * nodelist,
            unsigned char index )
```

destroy a given Node.

Removes Node from nodelist by setting the distanceToStart as 0.

Parameters

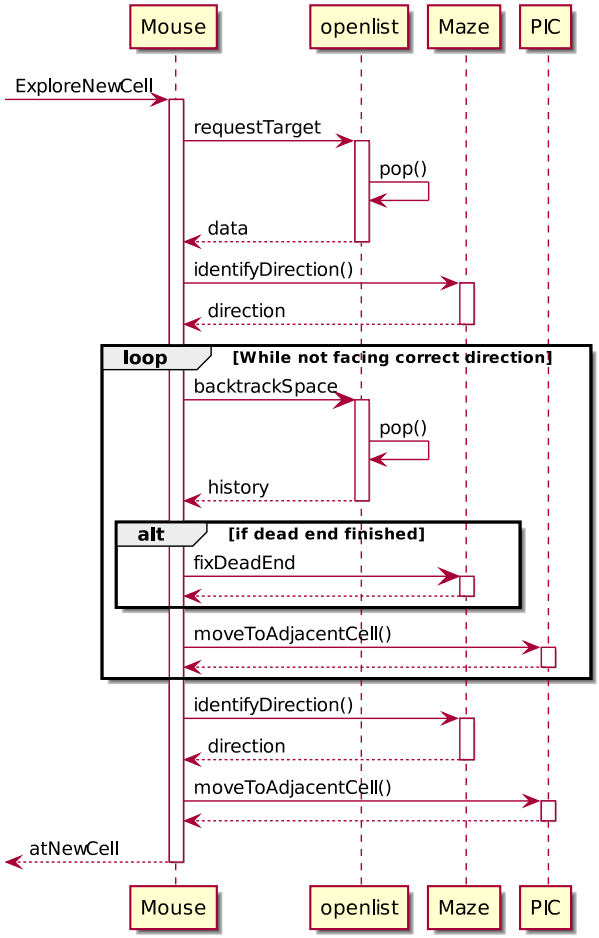
nodelist	List of all Nodes in the maze.
index	index where the Node to be destroyed is.

5.2.3.5 ExploreNewCell()

```
void ExploreNewCell (
            Stack * openlist,
            Stack * history,
            Node * nodelist )
```

Used to get to new areas.

Pops the first item from the openlist and explores the cell at that index. If the cell is not accessible from the current cell, then it backtracks until it finds it.



Parameters

openlist	The stack of cells to be explored.
history	stack of all the places visited by the mouse
nodelist	List of all the Nodes in the maze.

5.2.3.6 identifyDirection()

```
unsigned char identifyDirection (
            unsigned char target )
```

identify in which direction a cell is.

Identifies which direction an adjacent cell is in, if the target cell is not adjacent, then 0 is returned.

Parameters

target	the cell that the mouse is trying to get to.

Returns

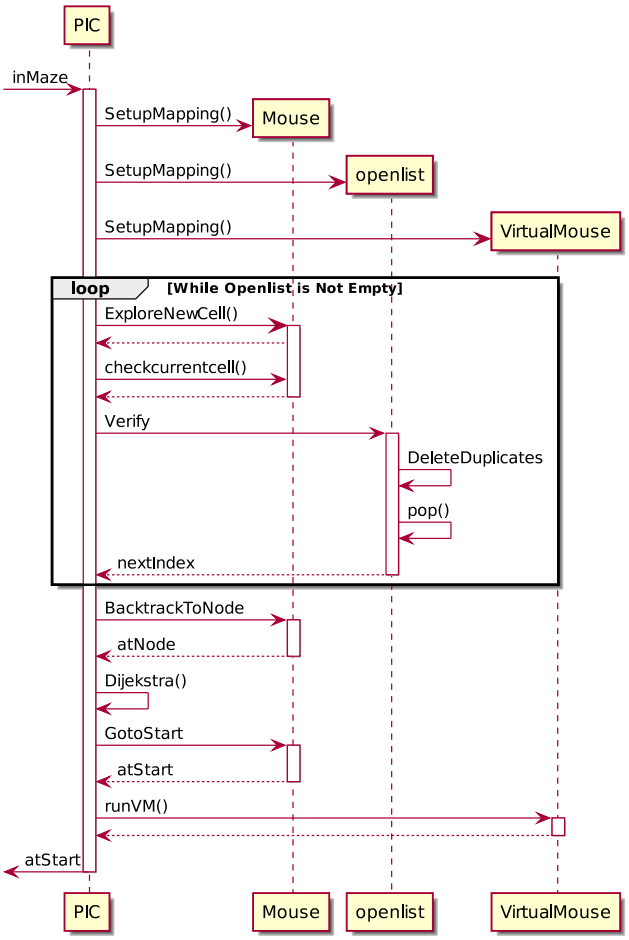
the direction of the adjacent cell.

5.2.3.7 mapmaze()

```
void mapmaze (
            struct Maze * mazeArg,
            Node * nodelist )
```

main maze-mapping function.

calls all the other functions to map out the whole reachable maze.



Parameters

mazeArg	pointer to the maze that will be populated by the mouse.
nodelist	list of all the Nodes in the maze. can be considered the Nodemap.

Returns

the index of the Node the mouse is currently at in the nodelist

5.2.3.8 moveToAdjacentCell()

```
void moveToAdjacentCell (
            unsigned char direction )
```

move mouse into an adjacent cell in the direction given.

Parameters

direction	the direction in which the adjacent cell is.

5.2.3.9 SetupMapping()

```
void SetupMapping (
            Stack * openlist,
            Node * nodelist )
```

sets up the mouse ready to map out the maze.

Initialises the mouse's maze model as all 0s with a border of 1s. Sets the mouse's index to 0 and direction to 0b1000 (North). Creates the start Node and seets it as the parent node.

Parameters

openlist	lise

5.2.3.10 virtualMouse()

```
void virtualMouse (
            Node * nodelist )
```

corrects any known but unexplored dead-ends.

Checks every cell in the maze, if the cell is unexplored and has 3 walls, then it will mark the cell as explored. This will get the cell removed from the openlist during the mapmaze function.

Parameters

nodelist	List of all the Nodes in the maze.

5.2.3.11 VMcheck()

```
void VMcheck (
            unsigned char index,
            Node * nodelist )
```

checks one cell for dead end and corrects it.

checks if cell is dead end by looking at wall pattern. If it is, it sets all the walls to 1, moves into the cell that connects to it and runs the check on that cell too. This means the "virtual mouse" will move back through the dead-end corridor, correcting it cell by cell, until it gets to either a non-fully-mapped cell or the end of that corridor.

Parameters

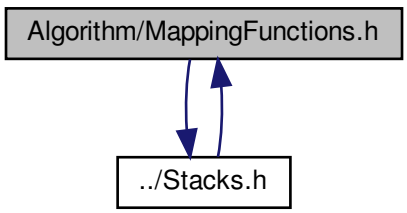
index	The index of the cell within the maze.
nodelist	List of all the Nodes in the maze.

5.3 Algorithm/MappingFunctions.h File Reference

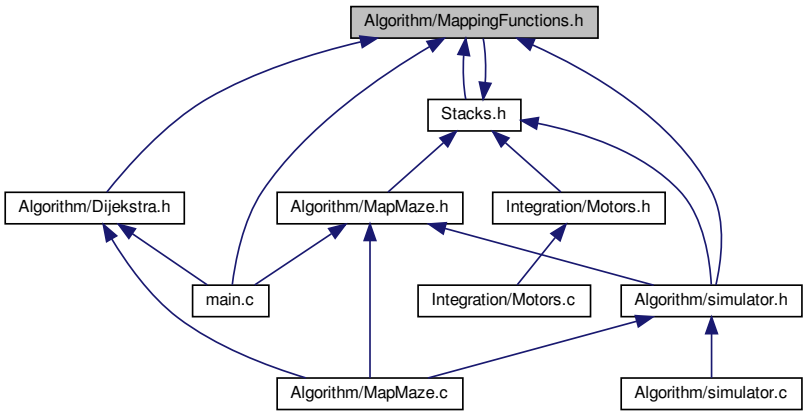
contains all functions, structures and data types used by most files.

```
#include "../Stacks.h"
```

Include dependency graph for MappingFunctions.h:



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



Classes

- struct connection

  Connection between 2 nodes.
- struct Node

  All info about a given Node.
- struct cell

  All info about a given cell.
- struct Maze

  Contains the representation of the maze itself.

Macros

- #define MAX_NODES 22

  maximum number of Nodes that can be stored.

Maze Max Dimensions

The maximum dimensions the maze can be.

This section will determine the size of most of the arrays and so should be set to the smallest values that will definitely contain the entire maze. If it is set to small then the mouse will not explore the whole maze.

- #define WIDTH 6
- #define HEIGHT 8
- #define MAX_DIMENSIONS

- #define TURN_COST 3

  cost of the movements between Nodes.
- #define **STRAIGHT_COST** 1

Typedefs

- typedef struct Node Node

  All info about a given Node.
- typedef struct cell cell

  All info about a given cell.

Functions

- unsigned char turn (unsigned char N, unsigned char dir)

  Turn the right mouse within the virtual maze.
- unsigned char incrementIndex (unsigned char index, unsigned char dir)

  Changes the index of the mouse to move into an adjacent cell.

5.3.1 Detailed Description

contains all functions, structures and data types used by most files.

Functions are used throughout the program, mostly to edit the maze information. Structures that are used to store info about the maze are defined in this header so can be accessed from any file that includes it. This will be necessary for almost all navigating of the maze.

Author

Nick Appleton @ UWE Robotics

Date

21/2/19

5.3.2 Macro Definition Documentation

5.3.2.1 HEIGHT

```
#define HEIGHT 8
```

height of the maze to be solved. If maze height is not known, set as largest value maze width that could occour.

5.3.2.2 MAX_DIMENSIONS

```
#define MAX_DIMENSIONS
```

marker to check if dimensions already defined.

5.3.2.3 MAX_NODES

```
#define MAX_NODES 22
```

maximum number of Nodes that can be stored.

describes the amount of memory needed to store the list of Nodes. Will not be able to store more Nodes than this.

5.3.2.4 WIDTH

```
#define WIDTH 6
```

Width of the maze to be solved. If maze width is not known, set as largest value maze width that could occour.

5.3.3 Function Documentation

5.3.3.1 incrementIndex()

```
unsigned char incrementIndex (
            unsigned char index,
            unsigned char dir )
```

Changes the index of the mouse to move into an adjacent cell.

looks at the direction the mouse is facing and changes the index by the right amount to move the mouse into the adjacent cell in that direction.

Parameters

index	index to be incremented.
dir	direction to move into.

Returns

the index having been incremented into the adjacent cell.

5.3.3.2 turn()

```
unsigned char turn (
            unsigned char N,
            unsigned char dir )
```

Turn the right mouse within the virtual maze.

Bitshifts the direction register of the mouse right by N places. Also corrects for if the 1 bit falls off the end of the register.

Parameters

N	Number of turns to make.
dir	Current direction to be turned.

Returns

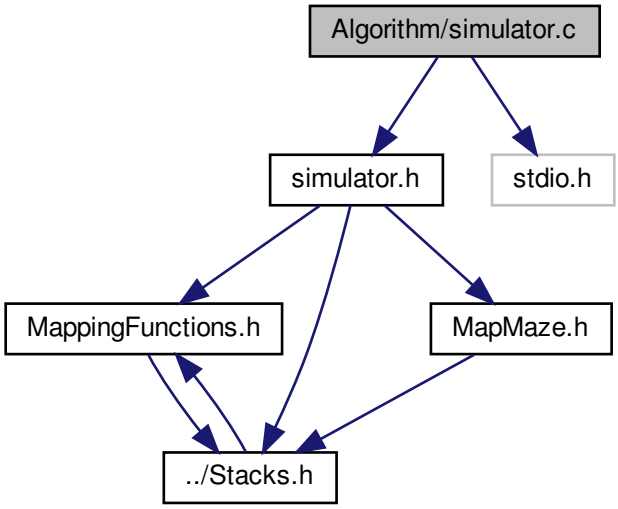
New direction after turning

5.4 Algorithm/simulator.c File Reference

Display the maze for debugging.

```
#include "simulator.h"
#include <stdio.h>
```

Include dependency graph for simulator.c:



Macros

Maze Actual Simulated Dimensions

The actual size of the simulated maze.

This section defines sizes that the simulated maze actually is. The maze is used in the readSensors function as all the other functions will refer to the mouses view of the maze.

- #define ACTUAL_WIDTH 6
- #define ACTUAL_HEIGHT 8

Functions

- int Wall_Check (unsigned char index, unsigned char direction)

  used to find if a wall is there or not.
- void printStatus (Mouse *mouse, Stack *openlist, Node *nodelist)

  Print out the maze to stdout.
- void Turn (int direciton)

  Turn physical Mouse.
- void Fwd_One_Cell (void)

  Move physical mouse forward one cell.
- void **Fast_Run** (Stack route, int speed)
- void **FastTurn** (int dir)

5.4.1 Detailed Description

Display the maze for debugging.

Adds functionality for the simulator to be used. This will mean that the program can be run without using the physical Micromouse. The functions have the same name as in the non- simulator functions so by defining SIMULATOR in the main it will run these instead of the actual mouse integratiion funtions.

Author

Nick Appleton @ UWE Robotics

Date

22/2/19

5.4.2 Macro Definition Documentation

5.4.2.1 ACTUAL_HEIGHT

```
#define ACTUAL_HEIGHT 8
```

the Height the simulated maze actually is

5.4.2.2 ACTUAL_WIDTH

```
#define ACTUAL_WIDTH 6
```

the Width the simulated maze actually is

5.4.3 Function Documentation

5.4.3.1 Fwd_One_Cell()

```
void Fwd_One_Cell (
            void  )
```

Move physical mouse forward one cell.

move the mouse forward one cell.

Dummy function as there is no physical mouse to turn. This is a placeholder so that the final program will run correctly on the simulator with minimal changing of code.

5.4.3.2 printStatus()

```
void printStatus (
            Mouse * mouse,
            Stack * openlist,
            Node * nodelist )
```

Print out the maze to stdout.

Prints the maze in it's current state to the standard output, including state of the LEDs, position and direction of the mouse and the mouse's model of the maze - this will display the walls and nodes graphically.

Parameters

mouse	representation of the mouse to be referenced.

5.4.3.3 Turn()

```
void Turn (
            int direciton )
```

Turn physical Mouse.

turn the mouse 90 degrees in a given direction.

Dummy function as there is no physical mouse to turn. This is a placeholder so that the final program will run correctly on the simulator with minimal changing of code.

Parameters

direciton	Direction to turn.

5.4.3.4 Wall_Check()

```
int Wall_Check (
```

```
            unsigned char index,
            unsigned char direction )
```

used to find if a wall is there or not.

contains the array of the actual maze walls and uses the index and direction of the mouse to determine if there is a wall or not. This is a simulated version of reading the sensor values and passing a 1 or 0 depending on the status of the wall.

Parameters

index	index that the mouse is at within the maze.
direction	direction register of the mouse.

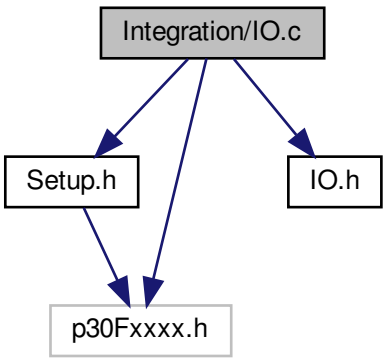
Which wall is being checked

5.5 Integration/IO.c File Reference

Functions for input and output Integration.

```
#include "Setup.h"
#include "IO.h"
#include <p30Fxxxx.h>
```

Include dependency graph for IO.c:



Macros

- #define noWall 10

- #define upperError 10

  thresholds for anomilous sensor values used in PID
- #define **lowerError** 10

Functions

- void __attribute__ ((interrupt, no_auto_psv))
  UART 2 receive interrupt for encoder 2 and USB interface.
- float Sensor_Read (int sensor)
  read the Sensors using the ADC.
- void Start_Position (void)
  Use LEDs to give feedback on position in cell.
- void Sensor_Test (void)
  check if there is a wall or not.
- void __attribute__ ((interrupt, auto_psv))
  Timer1 interrupt for LED display.
- void SetLED (unsigned char LED, unsigned char set)
  Turn LED on or off.
- void Display (unsigned char setmode, unsigned int speedchange)
  Set what is displayed on the LEDs.

Variables

- unsigned char sensorVal
  Value read by the sensor. Global because it is used in the interrupt.
- unsigned int speed
- unsigned char mode

5.5.1 Detailed Description

Functions for input and output Integration.

Includes the interrupts for sensor inputs and LED outputs as well as functions to decode values and make use of the IO

Author

Christian Woof @ UWE Robotics

5.5.2 Macro Definition Documentation

5.5.2.1 noWall

```
#define noWall 10
```

threshold for if a wall has been sensed.

5.5.3 Function Documentation

5.5.3.1 __attribute__() [1/2]

```
void __attribute__ (
             (interrupt, no_auto_psv)  )
```

UART 2 receive interrupt for encoder 2 and USB interface.

UART 1 receive interrupt for encoder 1 and programmer.

ADC interrupt for use with sensors.

5.5.3.2 __attribute__() [2/2]

```
void __attribute__ (
             (interrupt, auto_psv)  )
```

Timer1 interrupt for LED display.

reset the timer 1 interrupt flag

5.5.3.3 Display()

```
void Display (
             unsigned char setmode,
             unsigned int speedchange )
```

Set what is displayed on the LEDs.

Parameters

setmode	Which display Mode is to be used.
speedchange	Speed the display should go at.

5.5.3.4 Sensor_Read()

```
float Sensor_Read (
             int sensor )
```

read the Sensors using the ADC.

Parameters

sensor	which sensor to read from.

Returns

value between 0 and 255 of sensor read.

5.5.3.5 Sensor_Test()

```
void Sensor_Test (
            void  )
```

check if there is a wall or not.

Uses Sensor_Read() to get an average of the sensor values and uses this to determine whether there is a wall or not in the direciton of the sensor.

Parameters

side	which side of the mouse is being checked.

Returns

1 or 0 (Wall or no Wall). Test sensors

5.5.3.6 SetLED()

```
void SetLED (
            unsigned char LED,
            unsigned char set )
```

Turn LED on or off.

Parameters

LED	which LED is being set.
set	Set or Clear.

5.5.3.7 Start_Position()

```
void Start_Position (
            void  )
```

Use LEDs to give feedback on position in cell.

used to setup the mouse in the maze at the beginning by holding the mouse in the cell until the 2 LEDs turn off. This means that the mouse is in the correct position.

5.5.4 Variable Documentation

5.5.4.1 mode

```
unsigned char mode
```

LED mode to be displayed

5.5.4.2 speed

```
unsigned int speed
```

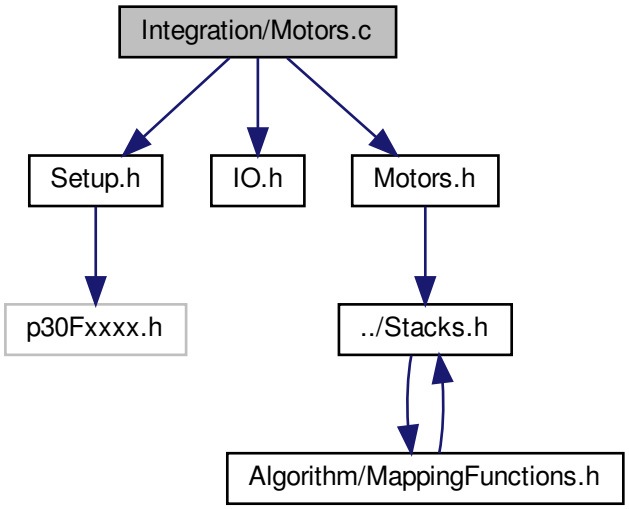
speed at which the LED patterns will be displayed.

5.6 Integration/Motors.c File Reference

Motor Functions and Defines.

```
#include "Setup.h"
#include "IO.h"
#include "Motors.h"
```

Include dependency graph for Motors.c:



Functions

- float PID (int close_Sensor_Side)

  PID controller for moving through the maze.
- void __attribute__ ((interrupt, no_auto_psv))

  ADC interrupt for use with sensors.
- void Velocity_Curve (unsigned char direction)

  accelerate the mouse up to speed, or from speed to static.
- void Turn_Velocity_Curve (unsigned char direction)

  accelerate or decelerate the mouse for the turns when exploring.
- void M_Dir (int MLDir, int MRDir)

  sets motor directions.
- void Turn (int turn)

  Turn physical Mouse.
- void Fwd_One_Cell (void)

  Move physical mouse forward one cell.
- void Fast_Run (Stack instructions, unsigned char speed)

  complete a fast run of a stack of instructions.
- void Fwd (int cells)

  move the mouse forward the number of cells given.
- void Fast_Turn (unsigned char dir)

  make a turn at full speed.

Variables

- int MLEncCount

  Encoder counts.
- int **MREncCount**

5.6.1 Detailed Description

Motor Functions and Defines.

Author

Christian Woof @ UWE Robotics

5.6.2 Function Documentation

5.6.2.1 __attribute__()

```
void __attribute__ (
            (interrupt, no_auto_psv)  )
```

ADC interrupt for use with sensors.

UART 1 receive interrupt for encoder 1 and programmer.

ADC interrupt for use with sensors.

5.6.2.2 Fast_Run()

```
void Fast_Run (
            Stack instructions,
            unsigned char speed )
```

complete a fast run of a stack of instructions.

follows the instructions in the given stack to get from the current location to the destination using the fast move functions. This will be used for the final run.

Parameters

instructions	Stack of instructions to follow to get to the destination.
speed	Whether it is going full or half speed.

5.6.2.3 Fast_Turn()

```
void Fast_Turn (
            unsigned char dir )
```

make a turn at full speed.

Parameters

dir	Direction to turn.

5.6.2.4 Fwd()

```
void Fwd (
            int cells )
```

move the mouse forward the number of cells given.

Parameters

cells	How many cells forward to move.

5.6.2.5 Fwd_One_Cell()

```
void Fwd_One_Cell (
             void  )
```

Move physical mouse forward one cell.

move the mouse forward one cell.

Dummy function as there is no physical mouse to turn. This is a placeholder so that the final program will run correctly on the simulator with minimal changing of code.

5.6.2.6 M_Dir()

```
void M_Dir (
             int MLDir,
             int MRDir )
```

sets motor directions.

1 for forward. 0 for stop. -1 for backwards.

Parameters

MLDir	Direction to set left motor to go.
MRDir	Direction to set right motor to go.

5.6.2.7 PID()

```
float PID (
             int close_Sensor_Side )
```

PID controller for moving through the maze.

Checks the sensor and slows down the opposite wheel if needed to keep the mouse in the centre of the maze.

Parameters

close_Sensor_Side	Side which sensor to be checked is on.

Returns

how much the opposite wheel needs to slow down by.

5.6.2.8 Turn()

```
void Turn (
            int direciton )
```

Turn physical Mouse.

turn the mouse 90 degrees in a given direction.

Dummy function as there is no physical mouse to turn. This is a placeholder so that the final program will run correctly on the simulator with minimal changing of code.

Parameters

direciton	Direction to turn.

5.6.2.9 Turn_Velocity_Curve()

```
void Turn_Velocity_Curve (
            unsigned char direction )
```

accelerate or decelerate the mouse for the turns when exploring.

Parameters

direction	whether accelerating (1) or decelerating (0).

5.6.2.10 Velocity_Curve()

```
void Velocity_Curve (
            unsigned char direction )
```

accelerate the mouse up to speed, or from speed to static.

Parameters

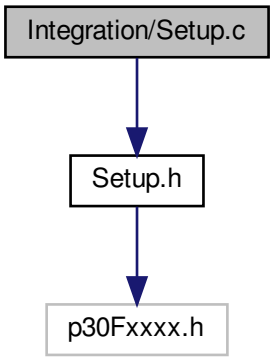
direction	whether accelerating (1) or decelerating (0).

5.7 Integration/Setup.c File Reference

Functions and Definitions for peripheral systems.

```
#include "Setup.h"
```

Include dependency graph for Setup.c:



Functions

- void IOSetup (void)
  IO Setup Function.
- void PWMSetup (void)
  PWM Setup Function.
- void UARTSetup (void)
  UART Setup Function.
- void ADC_Setup (void)
  ADC Setup Function.
- void **timer1Setup** (void)

5.7.1 Detailed Description

Functions and Definitions for peripheral systems.

Author

Christian Woof @ UWE Robotics

5.7.2 Function Documentation

5.7.2.1 ADC_Setup()

```
void ADC_Setup (
            void  )
```

ADC Setup Function.

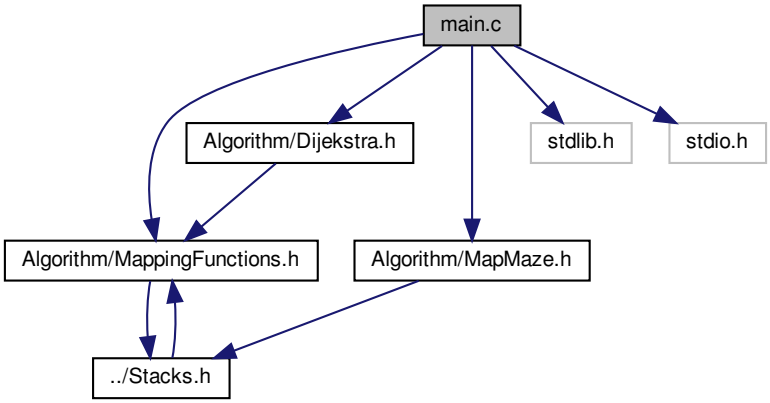
Timer 1 Setup Function.

5.8 main.c File Reference

this is the main file that controls the robot.

```
#include "Algorithm/MappingFunctions.h"
#include "Algorithm/MapMaze.h"
#include "Algorithm/Dijekstra.h"
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for main.c:



Functions

- **_FWDT** (WDT_OFF)
- int main (void)

  main.

5.8.1 Detailed Description

this is the main file that controls the robot.

Contains the mission planner and runs all initialisation of maze. offloads most functionality to external functions.

Author

Nick Appleton @ UWE Robotics

Date

21/2/19

5.8.2 Function Documentation

5.8.2.1 main()

```
int main (
            void  )
```

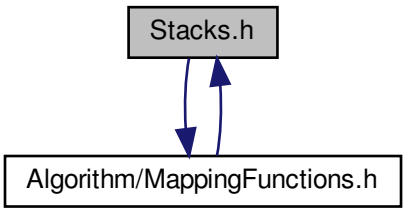
main.

Returns

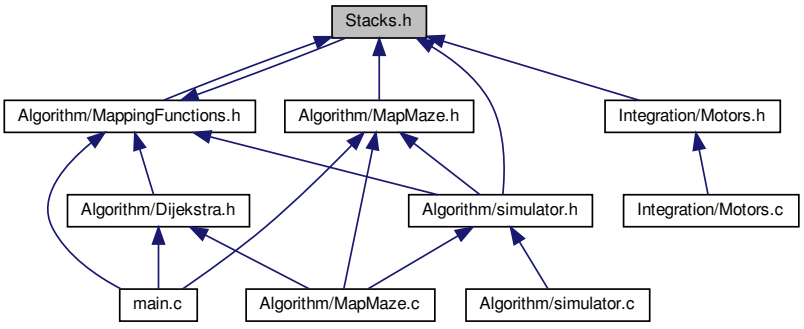
5.9 Stacks.h File Reference

Defines everything needed to implement stacks.

```
#include "Algorithm/MappingFunctions.h"
```
Include dependency graph for Stacks.h:



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



Classes

- struct Stack

  array of data that is the Stack.

Typedefs

- typedef struct Stack Stack

  array of data that is the Stack.

5.9.1 Detailed Description

Defines everything needed to implement stacks.

Includes the stack datatype, as well as the stackitem structure used to creat stacks. Stacks can have data pushed to them and popped from them. Uses Linked lists to do this.

Author

Nick Appleton @ UWE Robotics

Date

23/2/19

5.9.2 Typedef Documentation

5.9.2.1 Stack

```
typedef struct Stack Stack
```

array of data that is the Stack.

The size of the stack equal to number of cells in the maze.

Index