# Introduction

This document describes the capabilities exposes by the Motion Control Interface.

The various operation modes and their commands will be described in detail.

# Overview

The Squid robot control is managed by two distinct units:

-a mini computer (RPi3) running the SquidControl process under Linux

-a board build around a TI DSP (TI-Delfino F28377S)

Both will communication via a SPIO communication channel.

The robot can operate in the following ways:

* Manual mode: an operator controls the robot manually
* Autonomous mode: the robot operates autonomously according to a task received by the server

# Reference Document

|  |  |
| --- | --- |
| 1. **MotionControlICD.h** | ICD data structures. See this file for detailed field information. |
| 1. **Xbox controller commands and states.docx** | describes the Manual Mode state machine implemented in the robot first prototype. |

# Operation Modes

## Manual Mode

In the manual mode, the robot executes direct commands received from an operator using a controller.

The controller can change the steering mode and actuate three axes. Those are:

* Advancing speed
* Turn rate
* Wheel speed differential (used on horizontal track movement)

The state machine is described in the **Xbox controller commands and states.docx**.

In each state, the controls act as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Forw./Back axis | Left/Right axis | 3rd axis |
| GroundSteering | regular ground free steering | speed | Ackerman Steer |  |
| Fixed90DegSteering | wheels at 90 degrees, fixed. Used to move on vertical track | Speed |  | Speed offset of lower wheel |
| DifferentialSteering | wheels at 90 degrees, with differential steering. When 1 wheel turns right, the other turns left. Used to dock the track | speed | Parallel Steer | Speed offset of lower wheel |
| ParallelSteering | movement on rails, allow lower wheel differential speed | Speed | Parallel Steer | Speed offset of lower wheel |

Yahali comments

The TI controller receives from the RB functional commands.

The TI does not interact with any joystick, neither it has any direct interest in its controls.

The 0 or 90 degrees are not sacred by any means.

The parameters required by the robot are:

|  |  |  |
| --- | --- | --- |
| Parameter | Description | comment |
| Reference axis w.r.t. robot body (degrees) | Angle is zero when the robot's straight travel is directed towards its head  Positive angle is according to the right hand rule, z axis is down (nadir when robot is level) | This angle can only change when the robot is stationary. Changing it with the robot moving will be rejected with error  The angle will be set to +/-90 deg for right or left side crawl. |
| Line speed (m/sec) | Moving speed target. The robot will accelerate on its internally managed acceleration when attempting this speed. |  |
| Curvature (1/m) | Motion radius = 1/radius of curvature. Zero means a straight line. Positive curvature means turning to the positive (right hand rule) | Curvature will create difference between the moving wheels.  Entering too big curvature per the motion speed can put the robot to a skid – be careful. |

## Autonomous Mode

In this mode, the server sends a task to the robot. The task includes a route composed of segments and actions (e.g. pick/put item).

The SquidControl process then sends in due time the following types of commands to the Motion Control component:

* Navigation commands: segments information
* Actions: pick item, drop item, etc…
* Sensor information: current position, offset from indentified wanted position

The TI board reports the following:

* Position info (current segment, estimated position, velocity, azimuth, etc…)
* Acknowledgments for received commands
* Status: current operating mode, current action (e.g. moving, picking, etc…)
* Reports for completed actions (e.g. picked item number #)

Navigation segment types:

|  |
| --- |
| GroundStraight, //straight segment on the ground  GroundTurn, //ground turn Ackerman  DiffSpeedTurn, //First wheel forward second wheel backwards  TakeOff, //segment from closest ground path to terminal engaement  RailVertical, //movement on vertical rail  RailJunctionNoTurn, //junction crossing, this segment has only a position  RailJunctionWithTurn, //junction with change of direction  RailHorizontal, //movement on horizontal rail |

See **MotionControlICD.h** for details.

Note: The ground segments defined above ony allows for straight lines or circular arcs.

It is possible to allow for more general ground travel with the following additions:

- Bezier curves: may be complicated to measure and reproduce on the ground

-“follow ground line” mode: seen in various other systems. More involved controlled loop as the line identification algorithm runs in the RPi.

Yahali comments:

I don’t want straight, radi, and so on. Best use a Bezier – one parameterization covers them all and lot more. To be specific, use 3rd order B-Splines while x,y,and z share a parameter.

You don’t need to tell anything about junctions, rails and other.

Simply, a non-zero Z will tell the robot to take of, and a tripple knot may serve to break the trajectory with continuous position only, forcing a complete halt at the multiple knot.

Expected marks, like radio IDs, will always be related to a knot

Yahali comments more

In fact, the TI need not distinguish manual or auto mode. It is only that top level feedback information may be delivered or not.

Feedback information can be of one of the following types:

* At time xxx you deviated some distance to your right (left is negative) and your rotation was deviating (right hand rule) with some angle
* At the time xxx we were some distance before (ahead is negative) a given knot.

Yahali comments more:   
The robot should have a serial instruction lists, called queues.

The instructions are queued for execution.

Several queues will be available as a library, so you can build a new queue while one is executing.

You can append a new action to an executing queue, or you can terminate a queue. Termination specifies:

* Is the executing instruction to complete, or terminate immediately
* Who is the next queue (NULL queue, as a default, is "get stuck forever"

An instruction may be:

* Do a Bezier segment
* Set the **Reference axis w.r.t. robot body**
* Do something with the other servos (e.g. release air valve)
* Wait a given time (NULL is forever)

## ICD

All the messages are listed below:

|  |
| --- |
| //MSG DIRECTION: commands sent to motion control  CMD\_QUERY\_PROTOCOL\_VERSION, //PROTOCOL\_VERSION\_RECORD  CMD\_CONTROL\_MODE, //set to manual or autonomous control  CMD\_MANUAL\_STEERING\_STICK\_INPUT, //actual controller input, used only in manual mode  CMD\_UPLOAD\_NEXT\_AUTO\_SEGMENTS, //upload next segments used for autonomous navigation, may include current segment  CMD\_OFFSET\_INFO, //informs motion control of offset from expection marker (e.g. line)  CMD\_POSITION\_INFO, //exact position info with time tag  CMD\_PICK\_ITEM, //command to pick a specific item  CMD\_DROP\_ITEM, //command to drop loaded item, no message body  //MSG DIRECTION: reports sent by motion control  RPT\_PROTOCOL\_VERSION, //PROTOCOL\_VERSION\_RECORD  RPT\_CONTROL\_MODE, //report the actual control mode  RPT\_POSITION,  RPT\_PICK\_ITEM, //report of item picking status  RPT\_DROP\_ITEM, //report of item dropping status  //TODO still missing: errors, message acks |

Table 1ICD Messages

See **MotionControlICD.h** for the detailed fields of each message.