Tiberius Documentation

Release 1.0.0

Cameron A. Craig

May 11, 2016

CONTENTS

Contents:

CHAPTER

ONE

QUICKSTART

1.1 Tiberius - an introduction

Tiberius is the name of a robotics R&D platform developed by J.T Herd and his Masters students at Heriot-Watt University.

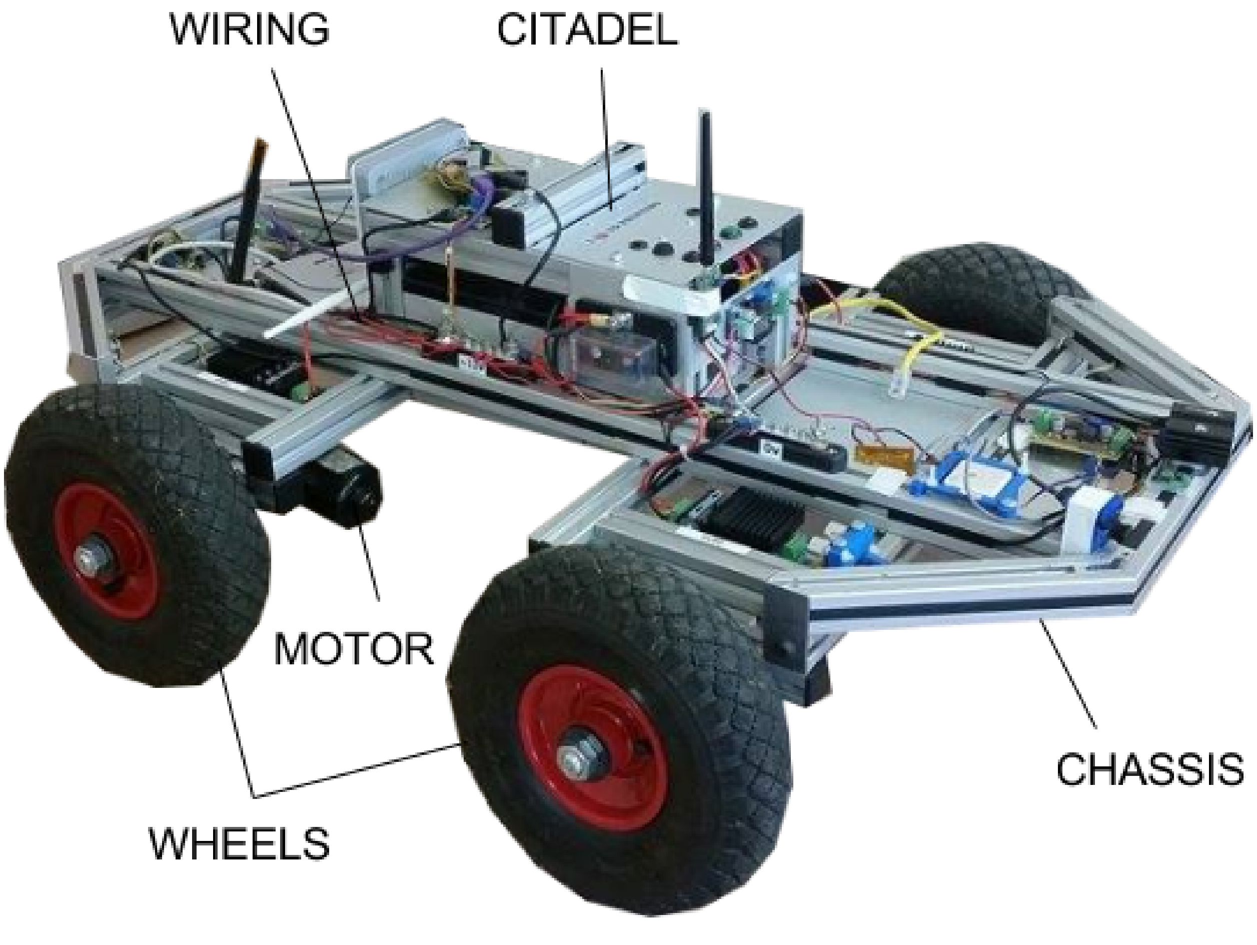

Fig. 1.1: Tiberius II Development Platform

1.1.1 Development Teams (2005 to Present)

The following section lists all known major contributors to the Tiberius project.

2015/16 Development Team

- Cameron Craig - Control API/Web Interface, IMDB
- Stuart Thain - Navigation Algorithms
- Euan Mutch - Communications, Robotic Arm Design
- Duncan Robertson - Suspension/Steering Design
- Andrew Rigg - Power Management
- Aidan Gallagher - SLAM (Simulataneous Localisation and Mapping)

2014/15 Development Team

- Sean Cunningham - Power System
- Tasos Deligiannis - Image Processing
- Dionysis Koutavas - Databases/Communication
- Denis Melehovs - Autonomy

1.2 Installation

The tiberius-robot package can be fully installed by running the setup script, found in the top level folder. The setup.py script attempts to install all necessary dependencies for tiberius-robot.

CHAPTER

TWO

TUTORIALS

2.1 Tutorials - an introduction

Please find the following tutorials useful. Each tutorial is designed to be as unambiguous as possible

2.1.1 Scope

These tutorials are designed to be read by developers. Whether they are part of the Tiberius development team at Heriot-Watt University, or an external developer interested in using, extending, forking, or developing the existing software repository.

2.2 Getting Started

The following tutorial will guide you through all the steps necessary to build a system using compatible hardware, and getting the system up and running. The instructions are ordered with the biggest steps at the start, so you don't get a head of yourself doing the easy bits at the beginning. In summary, in order to get a robot up and running you will need to set up a network, make sure you have compatible hardware, get a hold of our software, and configure our software for your hardware.

1. Installing compatible hardware

   The following pieces of hardware have been tested to work with the Tiberius Software Suite:

   - Raspberry Pi 1A+,1B,1B+,2B,3B
   - NEO-M8N (GPS Module, serial interface)
   - CMPS11 (Tilt Compensated Compass, I2C interface supported)
   - RPLIDAR (LIDAR, basic scan supported)
   - MD03 (Motor Driver, I2C interface supported)
   - RAMPS (3D Printer Driver Board, used to drive our robotic arm, serial interface)

   The exact model of DC motors used is not important, we have used wiper motors and motors from Como Drills.

   Our software assumes you'll have four driven wheels, any other number of wheels will require modification of the software. However this shouldn't be too difficult - modification of actuators.py to add or remove motors.

   Here two of our vehicles to get an idea of what we're talking about:

   A thorough hardware installation tutorial is available.

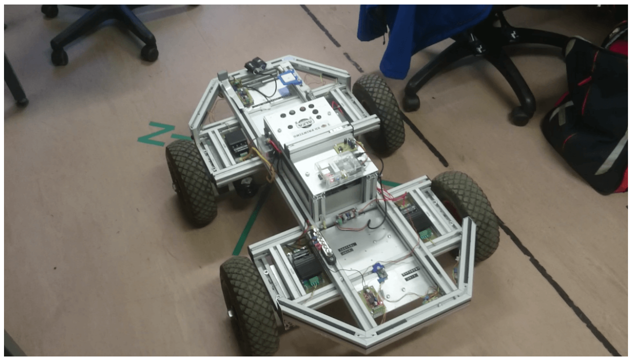

Fig. 2.1: Tiberius II

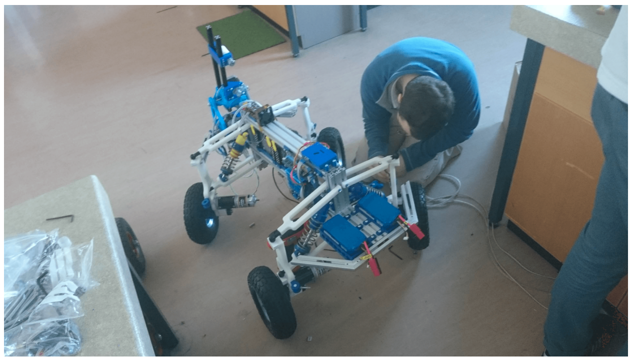

Fig. 2.2: Tiberius III

2. Getting a hold of the Tiberius software repository

   Once you have some hardware, you can think about getting a copy of our software. The way we would normally go about this is connecting the Raspberry Pi to the internet, through your LAN. The the separate tutorial (Networking Tutorial) for network configuration instructions.

   Note: It is *essential* to clone our git repository into into `/home/pi/git/`. This should result in the following folder being created:

   `/home/pi/git/tiberius-robot/`

   This is due to the hardcoded path names that are used in `setup.py`, in order to move files into a known location. Yes, we know, we know. This is not by any means ideal, but it is what currently works. This is an area that could be looked into if you feel like contributing.

   Here's an example of the commands that we use to clone our repository (assuming that you are currently in your home directory):

```
cd git/
git clone https://github.com/IonSystems/tiberius-robot.git
```

2. Installing the Tiberius Software Suite

   Now that you have the hardware and have cloned the repository, you can now think about installing our software.

Note: We have tried to make our setup script as thorough and reliable as possible, although we cannot guarantee success. Unless you have done some funny things to file permissions, it *should* work.

You should now locate yourself in the top level of our repository using `cd tiberius-robot/`. You can then run the setup script by typing `sudo python setup.py install`. They above commands are provided below for copy-paste convenience:

```
cd tiberius-robot/
sudo python setup.py install
```

3. Configuring your installation

   A number of configuration settings need to be edited to suit the particulars of your hardware setup. The configuration directory is the same for every installation, so your config file should appear in `/etc/tiberius/tiberius_conf.conf`

   It is important to ensure the correct hardware is enabled, and unavailable hardware is disabled. As this configuration file is used by the software to determine whether or not to communicate with the respective device. The configuration file is also used by the API to decide whether or not to allow access to the particular device through the API.

   It is also important to ensure the correct ports are set for the enabled devices. There is a test script to detect the USB devices available here: `tiberius/testing/scripts/gps_dev_path`. An alternative approach would be to run `start_tiberius.py` and you'll know if the ports are wrong if you see error messages for the device in question.

   Last, but not least, ensure the I2C addresses are set correctly. To list all I2C slaves on the bus, run `i2cdetect 1`. You'll need to work out the correspondence between the addresses and the devices by process of elimination, or by reading data sheets for default addresses.

4. Getting the software running

   This *should* be the easy part! We have a script in the top level of our repository called `start_tiberius.py`. This script takes care of starting everything in the correct order. If this starts successfully, then there is a good chance that everything is now operational.

   For a more in-depth discussion of what `start_tiberius.py` does, see docstrings.

2.3 Networking

2.4 Hardware Installation

CHAPTER

THREE

USER GUIDE

3.1 User Guides - an introduction

Tiberius the name of a robotics R&D platform developed by J.T Herd his Masters students at Heriot-Watt University.

3.2 Using the quickstart.py script

quickstart.py is located in the top level directory of this package, and provides shortcuts to common applications for convenience.

3.3 Tiberius's Database Module

3.3.1 Introduction

Tiberius's database module provides a consistent interface between your Python programs and a number of database technologies.

3.3.2 Installation

Installation of the database package is included in the setup script.

3.3.3 Tiberius's Database Interface

The database module currently supports the following SQL features: - INSERT - DELETE - DROP - UPDATE - SELECT

These SQL features are hidden behind the scenes in our wrapper classes, to give a consistent interface for multiple database technologies.

Example Usage

The following example shows all currently supported database interactions:

```
#Create a new Database
db = PolyhedraDatabase('example_db')

#Create a table with a few columns.
db.create('example_table', {'id': 'int primary key', 'example_column':'varchar(100)'})

#Insert a row into the table
db.insert('example_table', {'id': 0, 'example_column':'Example text value.'})

#Query the table for all rows an columns
results = db.query('example_table', '*')
print results

#Update the value we previously put in
db.update('example_table',
        {
            'example_column': 'Example new updates value.',
        },
        {
            'clause':'WHERE',
            'data': [
                {
                    'column' : 'id',
                    'assertion' : '=',
                    'value': '0'
                }
            ]
        })

#Drop the table
db.drop('example_table')
```

3.4 Tiberius's Architecture

3.4.1 Introduction

ROS (Robot Operating System) is an extremely popular development framework for robots. The publish-subscribe data model that is used by ROS works wonders, and rightly so no alternative data architecture has aver been considered. But, how would an alternative architecture compare? This is the question that we ask. And we hope to get part of the answer by implementing a super-fast in-memory database, and churning through all our data/messages. This will operate similar to the Blackboard (https://en.wikipedia.org/wiki/Blackboard_(design_pattern)) design pattern.

Our central database architecture will allow any device to access any piece of information, making it easy to integrate multiple devices, irrespective of the hardware or software it uses.

3.4.2 System Architecture Diagram

The following diagram illustrates an example system utilising our central database architecture:

Our database engine of choice is Polyhedra Lite, the free version of a commercial in-memory database. Although we have developed a Python wrapper class that creates a common Python interface between any database engine with minimal effort required.

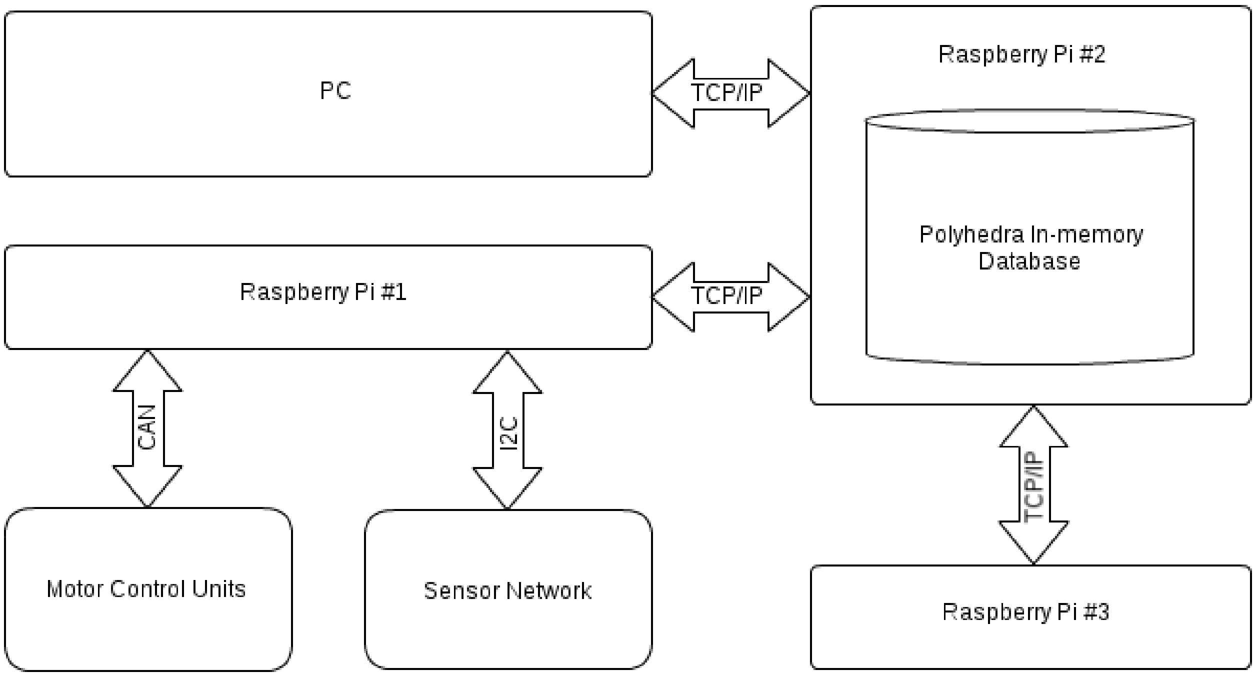

Fig. 3.1: Example System Architecture

3.4.3 Why not just use ROS?

This is a question that we asked ourselves when first presented with the Tiberius project. Well, you most definitely should use ROS, given the option. This project is motivated by the research interests of our project supervisor, Jim T. Herd. However, should you wish to explore a ROS alternative and be willing to to re-write a lot of Python to fit your hardware, then my all means, feel free.

3.5 Testing

3.5.1 Test Environment

Tiberius's test environment is designed to provide a consistent physical environment, where functional tests can be executed.

Note: The test environment and supporting functional tests are still in development, and not currently available for use.

3.5.2 Unit Testing

All new code *should* come with basic unit tests, although we don't guarantee this. All our unit tests are in the testing module. When creating unit tests we typically create all the test cases for the class/script/module in it's own file. Just have a look to see what I mean.

CHAPTER

FOUR

TIBERIUS

4.1 tiberius package

4.1.1 Subpackages

tiberius.database_wrapper package

Submodules

tiberius.database_wrapper.clauses module

class `tiberius.database_wrapper.clauses.`**`SqlClauses`**

A storage class containing SQL clauses, for use when building SQL statements.

`CREATE_TABLE` = **'CREATE TABLE'**

`DELETE` = **'DELETE'**

`DROP_TABLE` = **'DROP TABLE'**

`FROM` = **'FROM'**

`INSERT` = **'INSERT'**

`INTO` = **'INTO'**

`OR` = **'OR'**

`REPLACE` = **'REPLACE'**

`SELECT` = **'SELECT'**

`SET` = **'SET'**

`UPDATE` = **'UPDATE'**

`VALUES` = **'VALUES'**

`WHERE` = **'WHERE'**

tiberius.database_wrapper.database module

class `tiberius.database_wrapper.database.`**`Database`**

Bases: `object`

An abstract Database class. Subclasses of Database can be built for any database engine.

exception `DuplicateKeyError`(*value*)
Bases: `exceptions.Exception`

exception `Database.NoSuchTableError`(*value*)
Bases: `exceptions.Exception`

exception `Database.OperationalError`(*value*)
Bases: `exceptions.Exception`

exception `Database.TableAlreadyExistsError`(*value*)
Bases: `exceptions.Exception`

exception `Database.UnknownError`(*value*)
Bases: `exceptions.Exception`

`Database.create`(*table_name*, *columns*)

`Database.delete`()

`Database.drop`(*table_name*)

`Database.insert`(*table_name*, *values*)
A wrapper function allowing SQL insert statements to be generated without providing SQL statements directly.

Parameters

- **table_name** – The table to insert into.
- **values** – .

Returns A dictionary containing the results of the query.

`Database.query`(*table_name*, *column_name*, *conditions=None*)
A wrapper function allowing SQL quries to be generated without providing SQL statements directly.

Parameters

- **table_name** – Name of the table to query.
- **column_name** – The column name to return in the result of the query. Set to None to return all columns.
- **conditions** – A dictionary giving conditions for the query.

Returns A dictionary containing the results of the query.

Example

```
>>> import tiberius.database_wrapper.polyhedra_database as pd
>>> a = pd.PolyhedraDatabase("test_instance")
>>> a.sql("CREATE TABLE test_table (id int primary key, test_column varchar(1));")
>>> a.sql("INSERT INTO test_table (0, 'a');")
>>> a.query("test_table","test_column")
[('a', )]
>>> a.query("test_table","*")
[(0, 'a')]
```

`Database.update`()

tiberius.database_wrapper.polyhedra_database module

tiberius.database_wrapper.sqlite_database module

class `tiberius.database_wrapper.sqlite_database.`**`SqliteDatabase`**(*name*)

Bases: *`tiberius.database_wrapper.database.Database`*

`create`(*table_name*, *columns*)

`delete`(*table_name*, *conditions=None*)

`drop`(*table_name*)

`insert`(*table_name*, *values*)

`query`(*table_name*, *column_name*, *conditions=None*)

`set_db_name`(*name*)

`update`(*table_name*, *data*, *conditions*)

tiberius.database_wrapper.table module

class `tiberius.database_wrapper.table.`**`Table`**

Bases: `object`

`columns` = {}

`get_columns`()

Return a dict containing the column names and types. This dict can be directly passed into Database.create() in order to create a new table in the database.

`table_name` = ‘’

Module contents

4.1.2 Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX