# kobuki Documentation

***Release 1.0.0***

**Daniel Stonier**

**Nov 05, 2020**

# Kobuki


1 About 3

2 Out of the Box 5

3 Installing & Running the Software 7

4 Creating Applications 11

5 Troubleshooting 25

6 Specifications 29

7 Anatomy 31

8 Conversions 41

9 Serial Protocol 43

10 Firmware 55

11 Media 63

12 Docking Stations 65

13 Embedded Boards 67

14 Hardware Extensions 71

15 Non-C++ Kobuki 75

16 Documentation 77

17 Installation 79

18 ... 81

19 Changelog 83

20 Glossary 85

Index 87



iClebo Kobuki



iClebo Kobuki

CHAPTER 1

# About

Introducing Korea's first robotic turtle.

kobuki [] n. turtle

Kobuki is robotically engineered to be long-lived, tough and fast. With high performance batteries, Kobuki will tirelessly work alongside you through those long coffee-powered nights. He'll also happily burden himself with your modded array of sensors, actuators, laptops, embedded boards, portside cannons and do it all at a speed that makes his real world cousins seem like … well, turtles.

Use him for serving (chi-mek), chasing your neighbour's kids or simply, to make your own robot ideas become reality.

Kobuki is still young, don't expect him to remain as he is . Kobuki's development has already been significantly influenced by the community and as he marches towards old age, we will continue to work with the community and you to ensure he becomes better with time.

Sincerely, Kobuki Team.

CHAPTER 2

# Out of the Box

**Warning:** Be aware of the *Safety Guidelines*.

## 2.1 Charging

Connect the power adapter to Kobuki or dock Kobuki in the docking station. If Kobuki is turned on, you will hear a short sound when charging starts and the LED will light up appropriately.

| LED Color | Status |
|---|---|
| Green | fully charged |
| Blinking Green | charging |
| Orange | low battery |

**Note:** The battery still charges if Kobuki is off, but you will not see the LED, nor hear sounds

## 2.2 First Run

You want to see Kobuki in action without further ado? Kobuki has a special random walker mode embedded into the firmware which you can activate on start-up:

- Disconnect the power cable
- Turn on Kobuki.
- Within the first 3 seconds press the B0 button and hold for 2 seconds.
- LED2 will start blinking and Kobuki wander around.

**Note:** This was introduced to the firmware in v1.1.0. In case your kobuki is not running this or a later version, please refer to *Updating Firmware*.

CHAPTER 3

# Installing & Running the Software

## 3.1 Install from Binaries

If you happen to have access to a binary install (e.g. ROS), follow their instructions and then proceed directly to *Checking the Version Info*, otherwise follow the instructions below to build Kobuki and it's dependencies from source.

## 3.2 Build from Source

### 3.2.1 Requirements

The environment under which these instructions have been tested (and thus supported) is as follows.

- Platform: Linux (most flavours)
- C++ Version: c++14
- Compiler: gcc
- Build Dependencies: ament, colcon, vcstool, CMake
- Code Dependencies: Eigen, Sophus, ECL

Other platforms may work, but your mileage will vary. Windows has been supported in the past, so if you're willing to do a bit of work, you might find success.

### 3.2.2 Preparation

Ensure your system has the following packages installed:

- GCC (>=9)
- CMake (>=3.5)
- wget

- python3-venv

Download a few scripts that will help setup your workspace.

```
$ mkdir kobuki && cd kobuki

# a virtual environment launcher that will fetch build tools from pypi (colcon,␣
↪vcstools)
$ wget https://raw.githubusercontent.com/kobuki-base/kobuki_documentation/release/1.0.
↪x/resources/venv.bash || exit 1

# custom build configuration options for eigen, sophus
$ wget https://raw.githubusercontent.com/kobuki-base/kobuki_documentation/release/1.0.
↪x/resources/colcon.meta || exit 1

# list of repositories to git clone
$ wget https://raw.githubusercontent.com/kobuki-base/kobuki_documentation/release/1.0.
↪x/resources/kobuki_standalone.repos || exit 1
```

Fetch the sources:

```
$ source ./venv.bash

$ mkdir src

# vcs handles distributed fetching of repositories listed in a .repos file
$ vcs import ./src < kobuki_standalone.repos || exit 1

$ deactivate
```

**Note:** If you prefer to use your system Eigen:

```
$ touch src/eigen/AMENT_IGNORE
```

### 3.2.3 Build

```
$ source ./venv.bash

# build everything
$ colcon build --merge-install --cmake-args -DBUILD_TESTING=OFF

# disable any unused cmake variable warnings (e.g. sophus doesn't use BUILD_TESTING)
$ colcon build --merge-install --cmake-args -DBUILD_TESTING=OFF --no-warn-unused-cli

# build a single package
$ colcon build --merge-install --packages-select kobuki_core --cmake-args -DBUILD_
↪TESTING=OFF

# build everything, verbosely
$ VERBOSE=1 colcon build --merge-install --event-handlers console_direct+ --cmake-
↪args -DBUILD_TESTING=OFF

# build release with debug symbols
$ colcon build --merge-install --cmake-args -DBUILD_TESTING=OFF -DCMAKE_BUILD_
↪TYPE=RelWithDebInfo
```

(continues on next page)

(continued from previous page)

```
# update the source workspace
$ vcs pull ./src

$ deactivate
```

The resulting headers, libraries and resources can be found under `./install`.

These instructions are continuously vetted with a github action (yaml, results/logs).

## 3.3 Connect Kobuki

Kobuki's default means of communication is over usb (it can instead use the serial comm port directly, more on that later). On most linux systems, your Kobuki will appear on `/dev/ttyUSBO` as soon as you connect the cable. This is a typical serial2usb device port and if you happen to be using more than one such device, Kobuki may appear at `ttyUSB1`, `ttyUSB1`, ...

In order to provide a constant identifier for the connection, we've prepared a udev rule for you:

```
$ wget https://raw.githubusercontent.com/kobuki-base/kobuki_ftdi/devel/60-kobuki.rules
$ sudo cp 60-kobuki.rules /etc/udev/rules.d

# different linux distros may use a different service manager (this is Ubuntu's)
#   --> failing all else, a reboot will work
$ sudo service udev reload
$ sudo service udev restart
```

With this udev rule, you'll find your Kobuki appear at `/dev/kobuki` as soon as you connect and turn on the robot. This also comes with the added convenience that it is the default device port value for most Kobuki programs.

- Connect the usb cable
- Turn Kobuki on (you'll hear a chirp)
- Check for existence of `/dev/kobuki`
- I'm wearing a colander, you should too

If you're still having problems, refer to the *Troubleshooting* pages on *No USB Port / No Data*.

## 3.4 Checking the Version Info

```
# drop into the runtime enviroment
$ source ./install/setup.bash

# who is your kobuki?
$ kobuki-version-info
Version Info:
  Hardware Version: 1.0.4
  Firmware Version: 1.2.0
  Software Version: 1.1.0
  Unique Device ID: 97713968-842422349-1361404194
```

Your driver may give you a warning (software or firmware upgrade advised) or error (incompatible firmware/software) about mismatching versions. If it's the firmware you need to upgrade, refer to the section on *Firmware*.

## 3.5 Take Kobuki for a Test Drive

```
# drop into the runtime enviroment
$ source ./install/setup.bash

# take kobuki for a test drive
$ kobuki-simple-keyop
Simple Keyop : Utility for driving kobuki by keyboard.
KobukiManager : using linear  vel step [0.05].
KobukiManager : using linear  vel max  [1].
KobukiManager : using angular vel step [0.33].
KobukiManager : using angular vel max  [6.6].
Reading from keyboard
---------------------------
Forward/back arrows : linear velocity incr/decr.
Right/left arrows : angular velocity incr/decr.
Spacebar : reset linear/angular velocities.
q : quit.
current pose: [0, 0, 0]
current pose: [0, 0, 0]
current pose: [0, 0, 0]
current pose: [0.0064822, -1.17028e-06, -0.00074167]
current pose: [0.0226873, -9.88246e-05, -0.0133501]
```

CHAPTER 4

# Creating Applications

## 4.1 Chirp

### 4.1.1 About

This example merely configures and establishes a connection to Kobuki which will cause it to chirp, pause for five seconds and then emit the corresponding shutdown chirp. First though, some information about the library and the API that will be useful to get you started.

### 4.1.2 The Kobuki Library

The nature of the computational resources you have as well as your application's use case can have a significant impact on how you design your application, especially for details around the control loop. For this reason, the library does not endeavour to provide a control loop (that is up to you) and as such, libkobuki.so is simply one of classes, data structures, simple functions and collaback-oriented sigslot mechanisms.

### 4.1.3 The Kobuki Class

The kobuki:Kobuki class is the first port of call for developing your application. Configuration and non-callback modes of interaction are handled via this class. Callback modes are handled by sigslots, mroe on these later.

### 4.1.4 Initialisation & Configuration

Kobuki configuration is handled by the kobuki:Parameters class which is passed ot the kobuki instance via the kobuki::Kobuki::init() method. Most of the parameters to be configured have sane defaults.

The only one that requires frequent configuration is the serial device port. If you aren't using a udev rule to guarantee discovery at /dev/kobuki, then you'll typically find Kobuki at COM1 (windows) or /dev/ttyUSB0 (linux).

### 4.1.5 Code

```
#include <iostream>
#include <string>
#include <ecl/time.hpp>
#include <ecl/command_line.hpp>
#include <kobuki_core/kobuki.hpp>

class KobukiManager
{
public:
  KobukiManager(const std::string &device)
  {
    kobuki::Parameters parameters;
    // Specify the device port, default: /dev/kobuki
    parameters.device_port = device;

    // Other parameters are typically happy enough as defaults, some examples follow
    //
    // namespaces all sigslot connection names, default: /kobuki
    parameters.sigslots_namespace = "/kobuki";
    // Most use cases will bring their own smoothing algorithms, but if
    // you wish to utilise kobuki's minimal acceleration limiter, set to true
    parameters.enable_acceleration_limiter = false;
    // Adjust battery thresholds if your levels are significantly varying from␣
↪factory settings.
    // This will affect led status as well as triggering driver signals
    parameters.battery_capacity = 16.5;
    parameters.battery_low = 14.0;
    parameters.battery_dangerous = 13.2;

    // Initialise - exceptions are thrown if parameter validation or initialisation␣
↪fails.
    try
    {
      kobuki.init(parameters);
    }
    catch (ecl::StandardException &e)
    {
      std::cout << e.what();
    }
  }
private:
  kobuki::Kobuki kobuki;
};

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("chirp", ' ', "0.2");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  cmd_line.add(device_port);
```

(continues on next page)

(continued from previous page)

```
  cmd_line.parse(argc, argv);

  KobukiManager kobuki_manager(device_port.getValue());
  ecl::Sleep()(5);
  return 0;
}
```

## 4.2 Events & Streams

### 4.2.1 About

The next two applications make use of the callback handles provided by the core Kobuki class for listening in to events and streams from the Kobuki. This is done by registering callbacks with the sigslots framework.

### 4.2.2 Signals and Slots

The kobuki driver establishes a set of signals on uniquely labelled channels. Each channel consists of two parts. The first part represents the namespace, which can be customised via the sigslots_namespace variable in the kobuki::Parameter structure. The second uniquely identifies the signal itself.

The following represent the available signals along with the type of data they transmit when namespaced under the default namespace, "/kobuki".

**The Sensor Stream**

- /kobuki/stream_data [void]

The stream_data channel signals that a new data packet has arrived and is ready to be processed. These data packets are sent periodically and are include a composited payload containing data from all sensor streams. This is a special case, in that the type associated with the signal does not represent the data that has been collected, but just that it has arrived. This data can be fetched from within the callback connected to this signal via Kobuki::getCoreSensorData() which returns a kobuki::CoreSensors::Data structure holding all the important sensor information for the Kobuki.

**General Purpose Signals**

- /kobuki/ros_debug [std::string]
- /kobuki/ros_info [std::string]
- /kobuki/ros_warn [std::string]
- /kobuki/ros_error [std::string]
- /kobuki/version_info [kobuki::VersionInfo]: communicated only on request

**Event Handling Signals**

- /kobuki/button_event [kobuki::ButtonEvent]
- /kobuki/bumper_event [kobuki::BumperEvent]
- /kobuki/cliff_event [kobuki::CliffEvent]
- /kobuki/wheel_event [kobuki::WheelEvent]
- /kobuki/power_event [kobuki::PowerEvent]
- /kobuki/input_event [kobuki::InputEvent]

- /kobuki/robot_event [kobuki::RobotEvent]

These fire only when an event occurs.

Wheel events occur when the wheel position toggles between compressed and uncompressed (e.g. when you lift the robot from the floor). Input events correspond to gpio state changes (useful when you are customising Kobuki with additional sensors that can send binary signals to your program).

**Slots**

The kobuki driver does not establish any slots, that part is up to you and is demonstrated in the following program.

### 4.2.3 Code - Button Events

```
#include <iostream>
#include <random>
#include <string>
#include <vector>

#include <ecl/command_line.hpp>
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>

#include <kobuki_core/kobuki.hpp>

class KobukiManager
{
public:
  KobukiManager(const std::string &device) :
      slot_button_event(&KobukiManager::processButtonEvent, *this)
  {
    kobuki::Parameters parameters;
    parameters.device_port = device;

    try
    {
      kobuki.init(parameters);
    }
    catch (ecl::StandardException &e)
    {
      std::cout << e.what();
    }
    slot_button_event.connect("/kobuki/button_event");
  }

  /*
   * Nothing to do in the main thread, just put it to sleep
   */
  void spin()
  {
    ecl::Sleep sleep(1);
    while (true)
    {
      sleep();
    }
  }

  /*
```

(continues on next page)

(continued from previous page)

```
   * Catches button events and prints a curious message to stdout.
   */
  void processButtonEvent(const kobuki::ButtonEvent &event)
  {
    std::vector<std::string> quotes = {
      "That's right buddy, keep pressin' my buttons. See what happens!",
      "Anything less than immortality is a complete waste of time",
      "I can detect humour, you are just not funny",
      "I choose to believe ... what I was programmed to believe",
      "My story is a lot like yours, only more interesting 'cause it involves robots.
↪",
      "I wish you'd just tell me rather trying to engage my enthusiasm with these␣
↪buttons, because I haven't got one.",
    };
    std::random_device r;
    std::default_random_engine generator(r());
    std::uniform_int_distribution<int> distribution(0, 5);
    if (event.state == kobuki::ButtonEvent::Released ) {
      std::cout << quotes[distribution(generator)] << std::endl;
    }
  }

private:
  kobuki::Kobuki kobuki;
  ecl::Slot<const kobuki::ButtonEvent&> slot_button_event;
};

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("buttons", ' ', "0.1");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  cmd_line.add(device_port);
  cmd_line.parse(argc, argv);

  KobukiManager kobuki_manager(device_port.getValue());
  kobuki_manager.spin();
  return 0;
}
```

### 4.2.4 Code - The Sensor Stream

```
#include <iostream>
#include <string>

#include <ecl/command_line.hpp>
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>

#include <kobuki_core/kobuki.hpp>
```

(continues on next page)

(continued from previous page)

```
class KobukiManager
{
public:
  KobukiManager(const std::string &device) :
      slot_stream_data(&KobukiManager::processStreamData, *this)
  {
    kobuki::Parameters parameters;
    parameters.device_port = device;

    try
    {
      kobuki.init(parameters);
    }
    catch (ecl::StandardException &e)
    {
      std::cout << e.what();
    }
    slot_stream_data.connect("/kobuki/stream_data");
  }

  /*
   * Nothing to do in the main thread, just put it to sleep
   */
  void spin()
  {
    ecl::Sleep sleep(1);
    while (true)
    {
      sleep();
    }
  }

  /*
   * Called whenever the kobuki receives a data packet.
   * Up to you from here to process it.
   */
  void processStreamData()
  {
    kobuki::CoreSensors::Data data = kobuki.getCoreSensorData();
    std::cout << "Encoders [" << data.left_encoder << "," << data.right_encoder << "]
↪" << std::endl;
  }

private:
  kobuki::Kobuki kobuki;
  ecl::Slot<> slot_stream_data;
};

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("buttons", ' ', "0.1");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
```

(continues on next page)

(continued from previous page)

```
      "string"
  );
  cmd_line.add(device_port);
  cmd_line.parse(argc, argv);

  KobukiManager kobuki_manager(device_port.getValue());
  kobuki_manager.spin();
  return 0;
}
```

## 4.3 A Simple Control Loop

### 4.3.1 About

This example demonstrates how to process kobuki's pose data and based on the current pose, computes and sends the appropriate wheel commands to the robot, i.e. it closes the loop between sensing and control.

### 4.3.2 Code

Engage and watch Kobuki move around a dead-reckoned square with sides of length 1.0m.

```
#include <string>
#include <csignal>
#include <ecl/geometry.hpp>
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>
#include <ecl/linear_algebra.hpp>
#include <ecl/command_line.hpp>
#include "kobuki_core/kobuki.hpp"


/*****************************************************************************
** Classes
*****************************************************************************/

class KobukiManager {
public:
  KobukiManager(
      const std::string & device,
      const double &length,
      const bool &disable_smoothing
  ) :
    dx(0.0), dth(0.0),
    length(length),
    slot_stream_data(&KobukiManager::processStreamData, *this)
  {
    kobuki::Parameters parameters;
    parameters.sigslots_namespace = "/kobuki";
    parameters.device_port = device;
    parameters.enable_acceleration_limiter = !disable_smoothing;

    kobuki.init(parameters);
```

(continues on next page)

(continued from previous page)

```
    kobuki.enable();
    slot_stream_data.connect("/kobuki/stream_data");
  }

  ~KobukiManager() {
    kobuki.setBaseControl(0,0); // linear_velocity, angular_velocity in (m/s), (rad/s)
    kobuki.disable();
  }

  void processStreamData() {
    ecl::linear_algebra::Vector3d pose_update;
    ecl::linear_algebra::Vector3d pose_update_rates;
    kobuki.updateOdometry(pose_update, pose_update_rates);
    ecl::concatenate_poses(pose, pose_update);
    dx += pose_update[0];   // x
    dth += pose_update[2];  // heading
    // std::cout << dx << ", " << dth << std::endl;
    // std::cout << kobuki.getHeading() << ", " << pose.heading() << std::endl;
    // std::cout << "[" << pose[0] << ", " << pose.y() << ", " << pose.heading() << "]
↪" << std::endl;
    processMotion();
  }

  // Generate square motion
  void processMotion() {
    const double buffer = 0.05;
    double longitudinal_velocity = 0.0;
    double rotational_velocity = 0.0;
    if (dx >= (length) && dth >= ecl::pi/2.0) {
      std::cout << "[Z] ";
      dx=0.0; dth=0.0;
    } else if (dx >= (length + buffer)) {
      std::cout << "[R] ";
      rotational_velocity = 1.1;
    } else {
      std::cout << "[L] ";
      longitudinal_velocity = 0.3;
    }
    std::cout << "[dx: " << dx << "][dth: " << dth << "][" << pose[0] << ", " <<␣
↪pose[1] << ", " << pose[2] << "]" << std::endl;
    kobuki.setBaseControl(longitudinal_velocity, rotational_velocity);
  }

  const ecl::linear_algebra::Vector3d& getPose() {
    return pose;
  }

private:
  double dx, dth;
  const double length;
  ecl::linear_algebra::Vector3d pose;  // x, y, heading
  kobuki::Kobuki kobuki;
  ecl::Slot<> slot_stream_data;
};

/*****************************************************************************
** Signal Handler
```

(continues on next page)

(continued from previous page)

```
*****************************************************************************/

bool shutdown_req = false;
void signalHandler(int /* signum */) {
  shutdown_req = true;
}

/*****************************************************************************
** Main
*****************************************************************************/

int main(int argc, char** argv)
{
  ecl::CmdLine cmd_line("Uses a simple control loop to move Kobuki around a dead-
↪reckoned square with sides of length 1.0m", ' ', "0.2");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  ecl::ValueArg<double> length(
      "l", "length",
      "traverse square with sides of this size in length (m)",
      false,
      0.25,
      "double"
  );
  ecl::SwitchArg disable_smoothing(
      "d", "disable_smoothing",
      "Disable the acceleration limiter (smoothens velocity)",
      false
  );

  cmd_line.add(device_port);
  cmd_line.add(length);
  cmd_line.add(disable_smoothing);
  cmd_line.parse(argc, argv);

  signal(SIGINT, signalHandler);

  std::cout << "Demo : Example of simple control loop." << std::endl;
  KobukiManager kobuki_manager(
      device_port.getValue(),
      length.getValue(),
      disable_smoothing.getValue()
  );

  ecl::Sleep sleep(1);
  ecl::linear_algebra::Vector3d pose;  // x, y, heading
  try {
    while (!shutdown_req){
      sleep();
      pose = kobuki_manager.getPose();
      // std::cout << "current pose: [" << pose[0] << ", " << pose[1] << ", " <<␣
↪pose[2] << "]" << std::endl;
```

(continues on next page)

(continued from previous page)

```
    }
  } catch ( ecl::StandardException &e ) {
    std::cout << e.what();
  }
  return 0;
}
```

### 4.3.3 Decoupling the Control

This program relied on the periodic sensor stream to trigger the control commands. This results in a loop with the fewest lines of code as well as minimum latency between pose update and control.

Alternatively, you may wish to decopule the control from the sensor stream callback (e.g. via the spin() method). That is also fine and usual in more complex use cases. Beware however, of concurrency issues if using a separate thread.

## 4.4 Logging

### 4.4.1 About

Kobuki provides loggers over the debug, info, warning and error signals. By default, the software wires up stdout loggers directly to the warning and error signals, but you can both change this log level (e.g. DEBUG will cause all log levels to be printed to stdout) OR disable them complately and wire up slots to your own loggers.

### 4.4.2 Code - Log Levels

```
#include <iostream>
#include <string>
#include <ecl/console.hpp>
#include <ecl/time.hpp>
#include <ecl/command_line.hpp>
#include <kobuki_core/kobuki.hpp>

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("log_levels", ' ', "0.1");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  cmd_line.add(device_port);
  cmd_line.parse(argc, argv);

  std::cout << ecl::bold << "\nLog Levels Demo\n" << ecl::reset << std::endl;

  kobuki::Parameters parameters;
  parameters.device_port = device_port.getValue();
  parameters.log_level = kobuki::LogLevel::DEBUG;
```

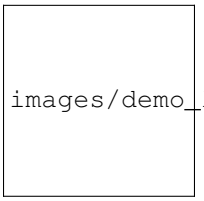
(continues on next page)

(continued from previous page)

```
  kobuki::Kobuki kobuki;
  try {
    kobuki.init(parameters);
  } catch (ecl::StandardException &e) {
    std::cout << e.what();
  }

  ecl::Sleep()(5);
  return 0;
}
```

### 4.4.3 Output - Log Levels




### 4.4.4 Code - Custom Loggers

```
#include <iostream>
#include <string>
#include <ecl/console.hpp>
#include <ecl/sigslots.hpp>
#include <ecl/time.hpp>
#include <ecl/command_line.hpp>
#include <kobuki_core/kobuki.hpp>

class KobukiManager
{
public:
  KobukiManager(const std::string &device) :
    slot_debug(&KobukiManager::logCustomDebug, *this),
    slot_info(&KobukiManager::logCustomInfo, *this),
    slot_warning(&KobukiManager::logCustomWarning, *this),
    slot_error(&KobukiManager::logCustomError, *this)
  {
    kobuki::Parameters parameters;

    parameters.device_port = device;
    // Disable the default loggers
    parameters.log_level = kobuki::LogLevel::NONE;

    // Wire them up ourselves
    slot_debug.connect(parameters.sigslots_namespace + "/debug");
    slot_info.connect(parameters.sigslots_namespace + "/info");
    slot_warning.connect(parameters.sigslots_namespace + "/warning");
    slot_error.connect(parameters.sigslots_namespace + "/error");

    try {
```

(continues on next page)

(continued from previous page)

```
      kobuki.init(parameters);
    } catch (ecl::StandardException &e) {
      std::cout << e.what();
    }
  }

  void logCustomDebug(const std::string& message) {
    std::cout << ecl::green << "[DEBUG_WITH_COLANDERS] " << message << ecl::reset <<␣
↪std::endl;
  }

  void logCustomInfo(const std::string& message) {
    std::cout << "[INFO_WITH_COLANDERS] " << message << ecl::reset << std::endl;
  }

  void logCustomWarning(const std::string& message) {
    std::cout << ecl::yellow << "[WARNING_WITH_COLANDERS] " << message << ecl::reset <
↪< std::endl;
  }

  void logCustomError(const std::string& message) {
    std::cout << ecl::red << "[ERROR_WITH_COLANDERS] " << message << ecl::reset <<␣
↪std::endl;
  }

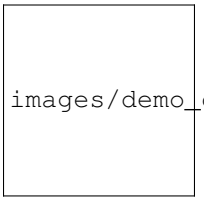
private:
  kobuki::Kobuki kobuki;
  ecl::Slot<const std::string&> slot_debug, slot_info, slot_warning, slot_error;
};

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("logging", ' ', "0.3");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  cmd_line.add(device_port);
  cmd_line.parse(argc, argv);

  std::cout << ecl::bold << "\nLogging Demo\n" << ecl::reset << std::endl;

  KobukiManager kobuki_manager(device_port.getValue());
  ecl::Sleep()(5);
  return 0;
}
```

### 4.4.5 Output - Custom Loggers




## 4.5 Debugging the Stream

### 4.5.1 About

If you're having troubles with your connection and need to debug the raw data stream, tune into the /kobuki/raw_data_stream signal.

### 4.5.2 Code

```
#include <iostream>
#include <string>
#include <ecl/console.hpp>
#include <ecl/sigslots.hpp>
#include <ecl/time.hpp>
#include <ecl/command_line.hpp>
#include <kobuki_core/kobuki.hpp>

class KobukiManager
{
public:
  KobukiManager(const std::string &device) :
    slot_raw_data_stream(&KobukiManager::logRawDataStream, *this)
  {
    kobuki::Parameters parameters;

    parameters.device_port = device;

    slot_raw_data_stream.connect(parameters.sigslots_namespace + "/raw_data_stream");

    try {
      kobuki.init(parameters);
    } catch (ecl::StandardException &e) {
      std::cout << e.what();
    }
  }

  void logRawDataStream(kobuki::PacketFinder::BufferType& buffer) {
    std::ostringstream ostream;
    ostream << ecl::cyan << "[" << ecl::TimeStamp() << "] " << ecl::yellow;
    ostream << std::setfill('0') << std::uppercase;
    for (unsigned int i = 0; i < buffer.size(); i++) {
      ostream << std::hex << std::setw(2) << static_cast<unsigned int>(buffer[i]) <<
↪" " << std::dec;
```

(continues on next page)

(continued from previous page)

```
    }
    ostream << ecl::reset;
    std::cout << ostream.str() << std::endl;
  }

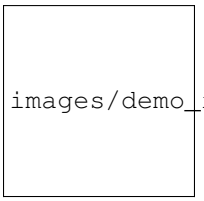
private:
  kobuki::Kobuki kobuki;
  ecl::Slot<kobuki::PacketFinder::BufferType&> slot_raw_data_stream;
};

int main(int argc, char **argv)
{
  ecl::CmdLine cmd_line("raw_data_stream", ' ', "0.3");
  ecl::ValueArg<std::string> device_port(
      "p", "port",
      "Path to device file of serial port to open",
      false,
      "/dev/kobuki",
      "string"
  );
  cmd_line.add(device_port);
  cmd_line.parse(argc, argv);

  std::cout << ecl::bold << "\nRaw Data Stream Demo\n" << ecl::reset << std::endl;

  KobukiManager kobuki_manager(device_port.getValue());
  ecl::Sleep()(5);
  return 0;
}
```

### 4.5.3 Output



CHAPTER 5

# Troubleshooting

## 5.1 No USB Port / No Data

Kobuki's FTDI chip is flashed with a special identifier that allows programs to uniquely identify the device as a kobuki. This in turn allows for udev rules that conveniently establish it's presence under /dev/kobuki.

### 5.1.1 Is it just Working?

Important checks that you can expect to see if and once it's working:

Does kobuki appear as USB device?

```
> lsusb
0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART) IC
```

Do you see it in dmesg when you insert the usb cable?

```
> dmesg
[  118.984126] usb 3-1: new full-speed USB device number 5 using xhci_hcd
[  119.139253] usb 3-1: New USB device found, idVendor=0403, idProduct=6001
[  119.139257] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[  119.139259] usb 3-1: Product: iClebo Kobuki
[  119.139261] usb 3-1: Manufacturer: Yujin Robot
[  119.139263] usb 3-1: SerialNumber: kobuki_A505QO28
[  119.150240] usbcore: registered new interface driver usbserial_generic
[  119.150249] usbserial: USB Serial support registered for generic
[  119.152383] usbcore: registered new interface driver ftdi_sio
[  119.152403] usbserial: USB Serial support registered for FTDI USB Serial Device
[  119.152505] ftdi_sio 3-1:1.0: FTDI USB Serial Device converter detected
[  119.152530] usb 3-1: Detected FT232RL
[  119.152665] usb 3-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

and when you remove it?

```
[  184.386124] usb 3-1: USB disconnect, device number 5
[  184.386507] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected␣
↪from ttyUSB0
[  184.386547] ftdi_sio 3-1:1.0: device disconnected
```

### 5.1.2 Problems & Solutions

- No /dev/kobuki?

```
# copy across udev rules
> sudo cp 60-kobuki.rules /etc/udev/rules.d
> sudo service udev reload
> sudo service udev restart
```

- Does kobuki stream data?

Check if anything is streaming - even when you don't have a program explicitly connected, you should see a stream of unusual characters.

```
> cat /dev/kobuki
```

If you don't have any streaming, check that your kernel has the ftdi_sio kernel driver built. Refer to kobuki_core#24 for more discussion.

- Everything seems fine, yet I still can't get the kobuki driver to communicate with it.

You may not be in the correct group, try the following and logout/login (or reboot):

```
> sudo addgroup $(USER) dialout
```

## 5.2 Unique Device ID?

Each Kobuki comes with a unique device ID imprinted on the FTDI chip at the factory. This can be retrieved with the kobuki-version-info program that comes as part of the kobuki_core package.

```
$ kobuki-version-info
Version Info:
  Hardware Version: 1.0.4
  Firmware Version: 1.2.0
  Software Version: 1.1.0
  Unique Device ID: 97713968-842422349-1361404194
```

If you need to engage with the company that you bought the Kobuki from, this is the number to report.

## 5.3 Version Mismatch

Your driver may give you a **warning** when it detects that your firmware's minor version is behind the latest supported by your driver:

```
Robot firmware is outdated; we suggest you to upgrade it
(hint: https://kobuki.readthedocs.io/en/devel/firmware.html)
Robot firmware version is 1.0.0, latest version is 1.2.0.
```

or **error** if a major version upgrade is required (usually indicative of a *Serial Protocol* change):

```
Robot firmware is outdated and needs to be upgraded
(hint: https://kobuki.readthedocs.io/en/devel/firmware.html)
Robot firmware version is 1.0.0, latest version is 1.2.0.
```

If this happens, then refer to the upgrade instructions in *Firmware*.

## 5.4 Malformed Payload

A malformed payload error occurs when Kobuki receives an unexpected byte or series of bytes in the long packets arriving on the serial connection. A typical error message will look something like:

```
[ERROR] Kobuki : malformed sub-payload detected. [225][170][E1 AA 55 4D 01 0F ]
[ERROR] Kobuki : malformed sub-payload detected. [42][170][2A AA 55 4D 01 0F ]
[ERROR] Kobuki : malformed sub-payload detected. [94][170][5E AA 55 4D 01 0F ]
[ERROR] Kobuki : malformed sub-payload detected. [63][170][3F AA 55 4D 01 0F C0 E8 00␣
↪00 00 ]
```

This is usually due to one of two causes:

1. Old or overly long cables
2. An FTDI driver configured with long latency

The first problem is easily diagnosed - simply try replacing cables (to be certain, ensure the cable length is under 2m).

The second problem is also easily diagnosed:

```
# Replace ttyUSB0 with ttyUSB# if it's not on the first port
$ cat /sys/bus/usb-serial/devices/ttyUSB0/tty/ttyUSB0/device/latency_timer
# If you see 16, your udev rule has not configured a non-default value (too slow!)
16
```

This was caused by a change in the kernel post the kobuki release which switched the default latency from 1ms to 16ms. As a result, the throughput is sub-optimal for Kobuki's use case. See kobuki#382 for more details (only if you're curious!).

The udev rules for Kobuki have already been updated to re-configure this latency for 1ms. If you're seeing 16ms, it means you haven't yet migrated to using the new udev rules.

Simply grab a copy of the new udev rule 60-kobuki.rules and:

```
# copy across udev rules
> sudo cp 60-kobuki.rules /etc/udev/rules.d
> sudo service udev reload
> sudo service udev restart
```

The key change is in the addition of a ATTR{device/latency_timer}="1" field in the rule.

CHAPTER 6

# Specifications

## 6.1 Safety Guidelines

- Do not twist or subject the power cable to extreme pressure or weight weight.
- Keep the pin and interface of the power plug clean from dust or water.
- Do not pull the power cord or touch the power plug with wet hands.
- Do not use with a damaged power plug, power cord or loose outlet.
- Use Kobuki indoors only.
- Do not pour or spray water onto Kobuki.
- Do not use Kobuki to pick up anything that is burning or smoking.
- Always remove the battery before long-term storage or transportation.

## 6.2 Functional

- Maximum translational velocity: 70 cm/s
- Maximum rotational velocity: 180 deg/s (>110 deg/s gyro performance will degrade)
- Payload: 5 kg (hard floor), 4 kg (carpet)
- Cliff: will not drive off a cliff with a depth greater than 5cm
- Threshold Climbing: climbs thresholds of 12 mm or lower
- Rug Climbing: climbs rugs of 12 mm or lower
- Expected Operating Time: 3/7 hours (small/large battery)
- Expected Charging Time: 1.5/2.6 hours (small/large battery)
- Docking: within a 2mx5m area in front of the docking station

## 6.3 Hardware

- PC Connection: USB / RS232 via RX/TX pins on the parallel port
- USB Serial Converter: FT232R chip from FTDI
- Motor Overload Detection: disables power on detecting high current (>3A)
- Odometry: 52 ticks/enc rev, 2578.33 ticks/wheel rev, 11.7 ticks/mm
- Gyro: factory calibrated, 1 axis (110 deg/s)
- Bumpers: left, center, right
- Cliff sensors: left, center, right
- Wheel drop sensor: left, right
- Power connectors: 5V/1A, 12V/1.5A, 12V/5A
- Expansion pins: 3.3V/1A, 5V/1A, 4 x analog in, 4 x digital in, 4 x digital out
- Audio : several programmable beep sequences
- Programmable LED: 2 x two-coloured LED
- State LED: 1 x two coloured LED [Green - high, Orange - low, Green & Blinking - charging]
- Buttons: 3 x touch buttons
- Battery: Lithium-Ion, 14.8V, 2200 mAh (4S1P - small), 4400 mAh (4S2P - large)
- Firmware upgradeable: via usb
- Sensor Data Rate: 50Hz
- Recharging Adapter: Input: 100-240V AC, 50/60Hz, 1.5A max; Output: 19V DC, 3.16A
- Netbook recharging connector (only enabled when robot is recharging): 19V/2.1A DC
- Docking IR Receiver: left, centre, right

## 6.4 Communication

- Baud rate: 115200 BPS, Data bit: 8 bit, Stop bit: 1 bit, No Parity

## 6.5 Software

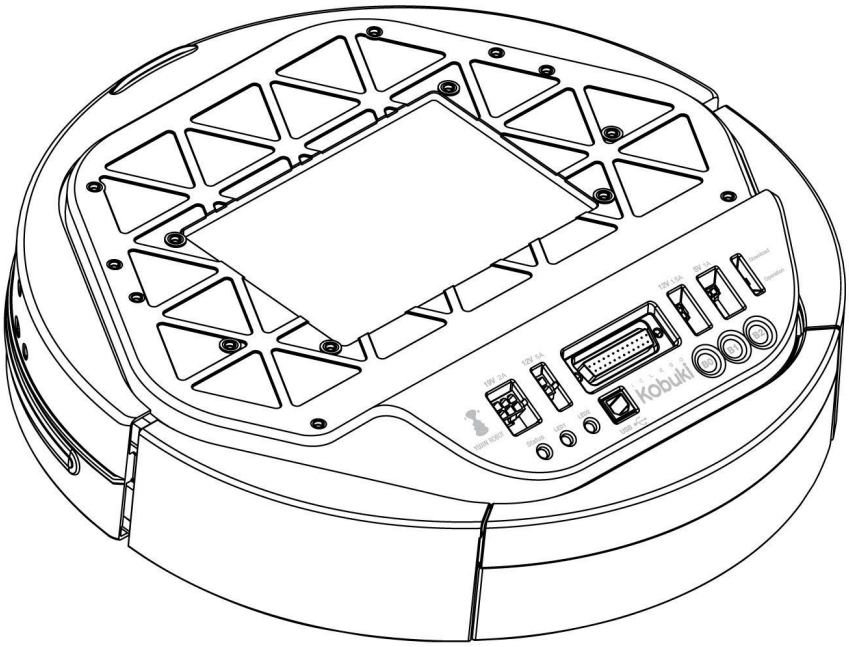
- C++ on Linux
- SDK, Version Checker, Simple Keyop, USB-FTDI utilities

CHAPTER 7

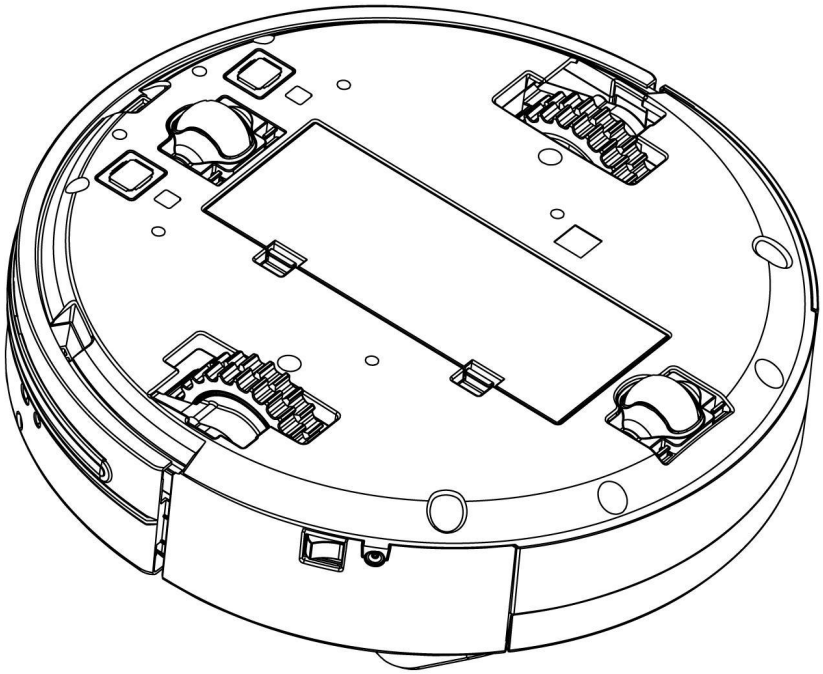
# Anatomy

## 7.1 Views

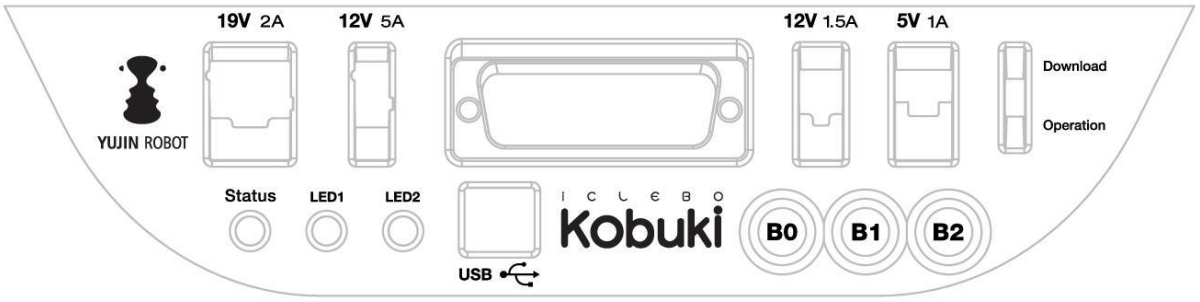
### 7.1.1 Top



### 7.1.2 Bottom



### 7.1.3 Control Panel




- 19V/2A: Laptop power supply
- 12V/5A: Arm power supply
- 12v/1.5A: Microsoft Kinect power supply
- 5V/1A: General power supply
- Status LED: Indicates Kobuki's status
- Green: Kobuki is turned on and battery at high voltage level
- Orange: On - Low battery voltage level (please charge soon)
- Green blinking: On - Battery charging

- Off: Kobuki is turned off.
- LED1/2: Programmable LEDs
- USB: Data connection
- BO/1/2: Buttons
- Firmware switch: Enable/disables the firmware update mode

## 7.2 Connectors

**Note:** SOME NOTES ABOUT THE MOLEX PAGES BELOW

1. We do not actually use Molex connectors but we are supplied by a Korean vendor who produces connectors according to the Molex standard. These links will be more useful to internationals in helping them find a mating part that works for them.
2. The images on each page are representative of the series of connectors. Each series usually has a variety of connectors with a different number of pins. As a result, the pictures on some of the pages below may seem as though they have the incorrect number of pins, but do not worry about this – they are the correct links. Note that you can jump to different connectors in the series via the second part of their identification number (e.g. 43045-0224 for the 2-pin, 43045-0424 for the 4-pin).
3. If some linked connectors are listed as obsolete on the molex website, don't worry. The connector you are exactly requiring are those you can find under the 'Mates with Parts' link on each page. If these however should become obsolete as well, please let us know via email.

### 7.2.1 Power




- 5V@1A Molex PN : 43650-0218 – for custom embedded boards (e.g. Arduino, Odroid)
- 12V@1.5A : Molex PN : 43045-0224 – for depth sensors (originally designed for Kinect/Asus sensors)
- 12V@5A : Molex PN : 3929-9023 – for high powered accessories (e.g. robotic arm)
- 19V@2A : Molex PN : 3928-9068 – for recharging netbooks (with a modified adapter)

### 7.2.2 Battery

- 4S1P/4S2P Battery Pack Connector: Molex PN : 00390-12040

### 7.2.3 I/O Port

DB25 pin D-SUB Female connector that provides the following functionality (pdf)

### 7.2.4 Cables

**Note:** If you click on the preceding links for the power connectors, under the heading 'Mates with Part(s)' you can find the compatible connector to use with each power source. The most important one being of course:

- 12V@1.5A : Molex PN : 43025-0200 – specially supporting the kinect

## 7.3 Models & Drawings

The models and drawings include both the base and parts for the Turtlebot 2.

- 2D mechanical drawings – DWG, PDF
- 3D models – IGS, STEP

The inserts in the kobuki plate are M4 threads (metric, 4mm). If you wish to build standoffs compatible for these inserts, please reference the pole pdf's in the 2D mechanical drawings which are what we use for turtlebots.

## 7.4 Motors

### 7.4.1 Specifications

- Brushed DC Motor
- Motor Manufacturer: Standard Motor
- Part Name: RP385-ST-2060
- Rated Voltage: 12 V
- Rated Load: 5 mN·m
- No Load Current: 210 mA
- No Load Speed: 9960 rpm $\pm$ 15%
- Rated Load Current: 750 mA
- Rated Load Speed: 8800 rpm $\pm$ 15%
- Armature Resistance: 1.5506 $\Omega$ at 25°C
- Armature Inductance: 1.51 mH
- Torque Constant(Kt): 10.913 mN·m/A
- Velocity Constant(Kv): 830 rpm/V
- Stall Current: 6.1 A
- Stall Torque: 33 mN·m

### 7.4.2 Control Method

- Driven by voltage source(H-bridge)
- Controlled by Pulse-width modulation(PWM)

## 7.5 Gyro

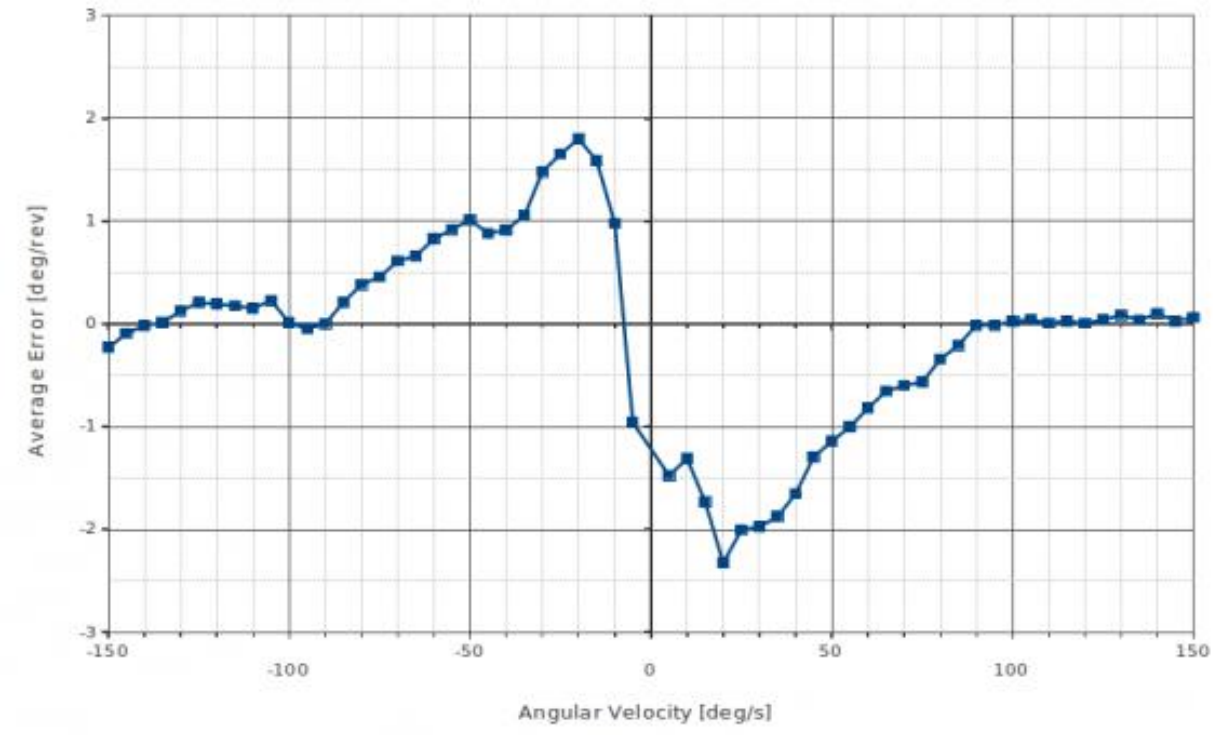
### 7.5.1 Specifications

- 3-Axis Digital Gyroscope
- Manufacturer : STMicroelectronics
- Part Name : L3G4200D
- Measurement Range: ±250 deg/s
- Yaw axis is factory calibrated within the range of ±20 deg/s to ±100 deg/s

### 7.5.2 Performance

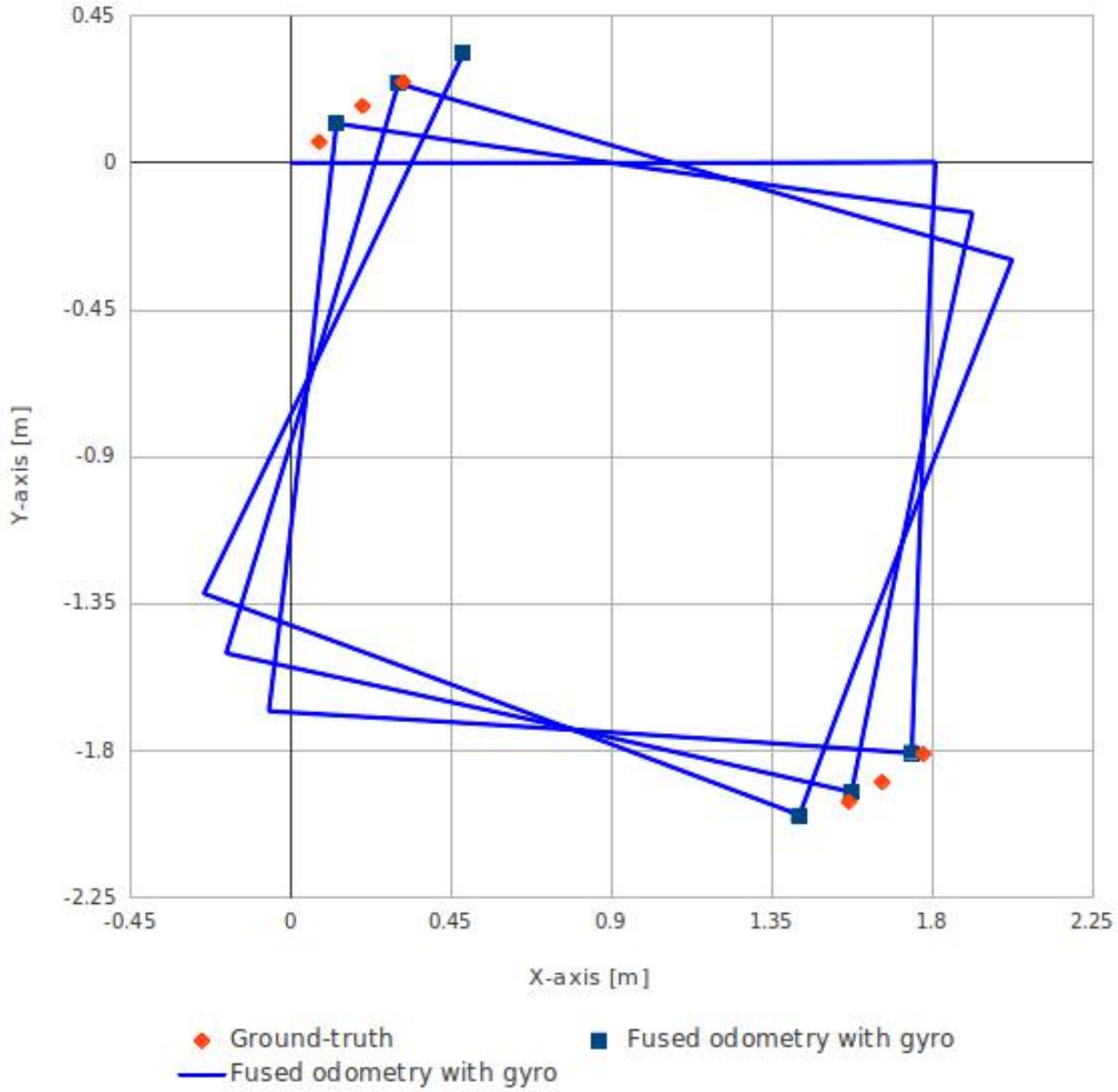
#### In-Place Rotation Test

This graph shows the average heading error per revolution of gyro, when robot rotates with a given velocity.



### Square Path Test

This graph shows the position error of fused odometry with gyro, when robot moves along a square path. Robot moved with 0.1 m/s on the line segment and rotated with 30 deg/s on the corner.




This table shows the calculated angular error, when robot arrived at the diagonally opposite corner from the starting point (0.0, 0.0).

| Number of turns of square path | Angular Error [deg] |
|---|---|
| 0.5 | 0.47 |
| 1.5 | 1.99 |
| 2.5 | 3.18 |

## 7.6 Power Adapter

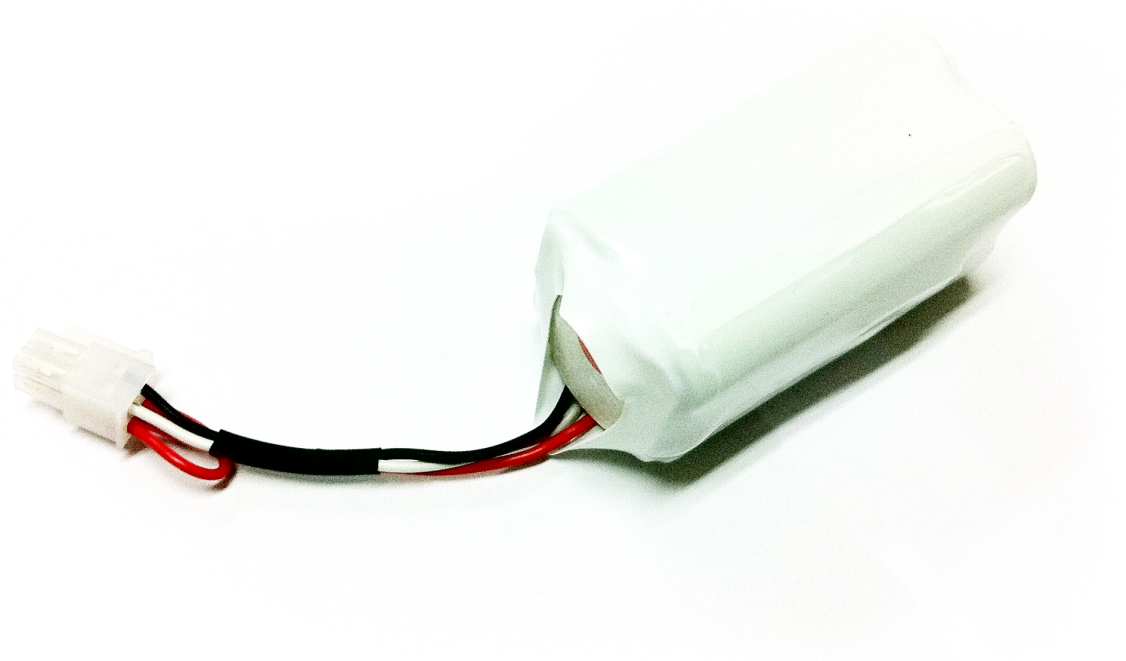
### 7.6.1 Specifications

| Input | Output |
|---|---|
| Voltage: 100-240V | Voltage: 19V |
| Ampere: 1.5A Max | Ampere: 3.16A |
| Frequency: 50/60Hz | Ampere: 3.16A |

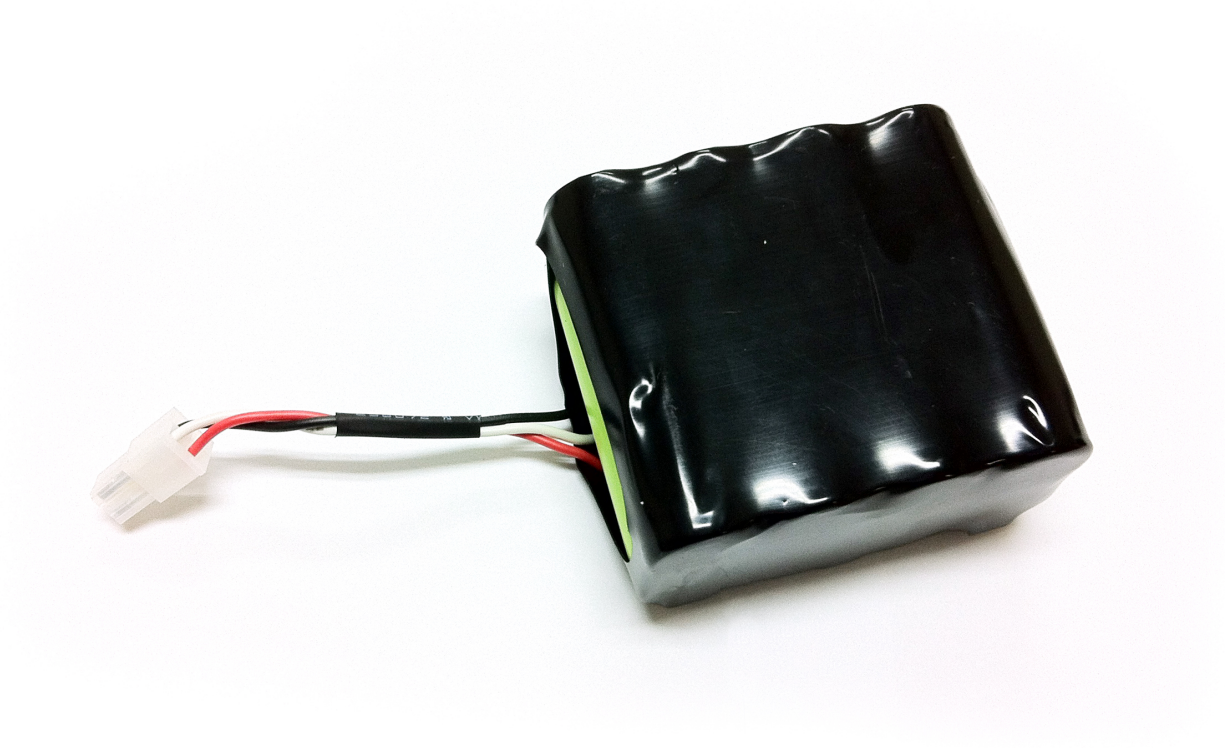
- Data Sheet - Charger (pdf).

## 7.7 Batteries

Kobuki by the default ships with a small Lithium-Ion battery pack (4S1P, 2200mAh, 14.8V).



**Tip:** For extra long operation, a big battery pack (4S2P, 4400mAh, 14,8V) can be ordered as well.



**Warning:** The electronics does not support the use of multiple battery packs at the same time (even if there is room in the battery compartment).

### 7.7.1 Specifications

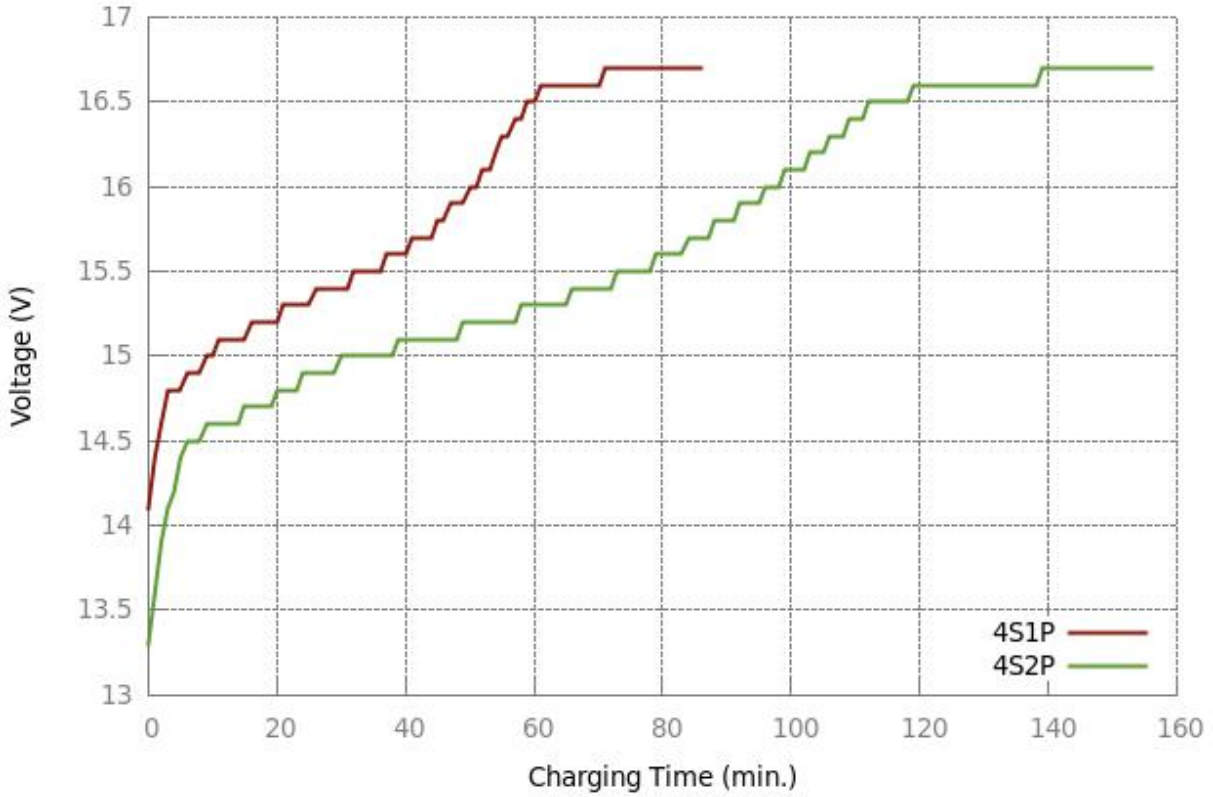
- Data Sheet - Battery Pack (pdf)

### 7.7.2 Pinouts

- Red : battery (+), 9.6 V ~ 16.8 V
- White: NTC thermistor to ground, 10 kΩ ± 1%
- Black: battery(-), Ground

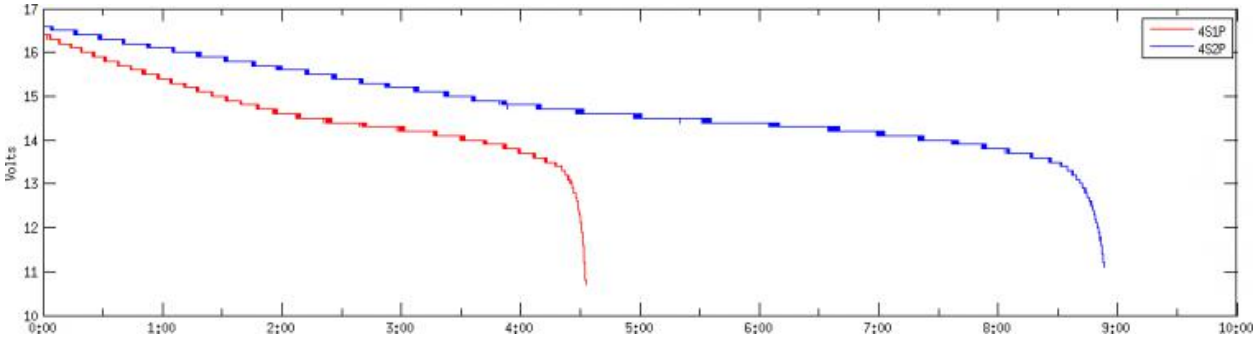
### 7.7.3 Charging Profile

This plot shows the voltages as measured by the robot's hardware. Both the standard 4S1P and the extra 4S2P batteries are compared. During the test, the robot was charging via adaptor.



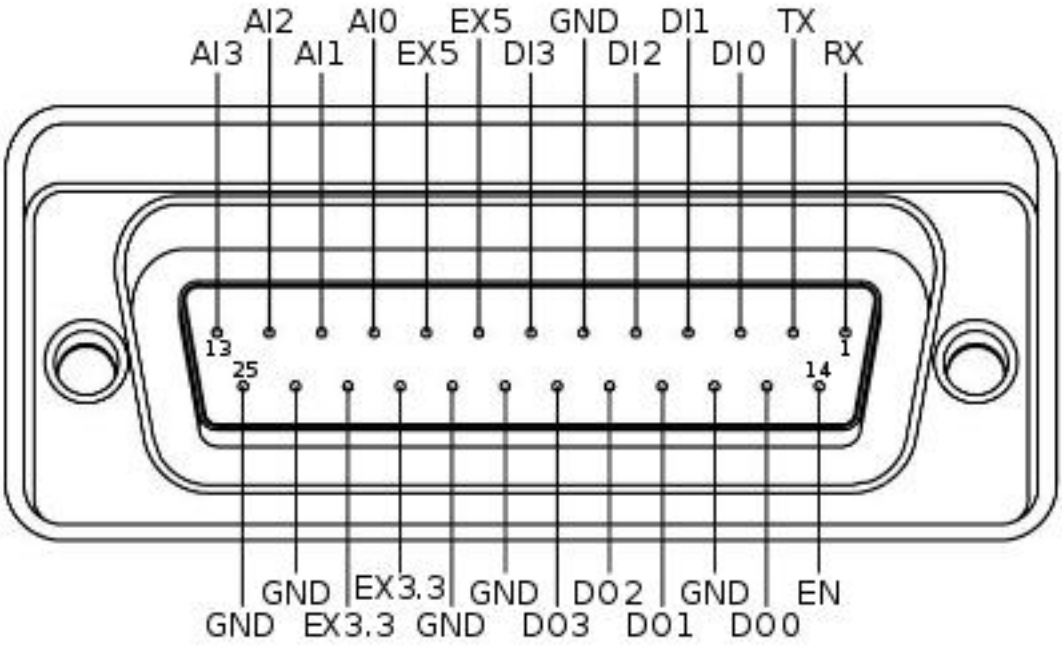

### 7.7.4 Discharging Profile

This plot shows the voltage as measured by the robot's hardware. Both the standard 4S1P and the extra 4S2P batteries are compared. During the test, the robot was continuously spinning, with the Kinect camera running.




## 7.8 Expansion Port

Pictured below are the pinouts of Kobuki's expansion port, including the serial pins. The minimum number of required pins for serial communication is three; TX, RX, and GND. Additionally EX3.3 or EX5 can be used for powering external devices, such as line transceiver.




- RX / TX: Serial data connection (RS232; used voltage level is 3.3V!)
- EX3.3 / EX5: 3.3V/1A and 5V/1A power supply
- DI0 - 3: 4 x Digital input (high: 3.3 - 5V, low: 0V)
- DO0 - 3: 4 x Digital output (open-drain, pull-up resistor required)
- AI0-3: 4 x Analog input (12bit ADC: 0 - 4095, 0 - 3.3V)
- GND: Ground
- EN: Used for detecting an external board (connect to external ground)

CHAPTER 8

# Conversions

## 8.1 Encoder2Pose

Here are the necessary parameters and calcualations for conversion of encoder ticks to robot pose.

| | Name | Value | Description |
|---|---|---|---|
| **Robot Parameters** | wheelbase (bias) | 230mm | length between the centre of the wheels |
| | wheel radius | 35mm | |
| | wheel width | 21mm | |
| **Magnetic Encoder** | ticks per revolution | 52 tick/rev | |
| | pulses per revolution | 13 pulse/rev | |
| **Gear Box** | 1st stage | 1:10 | |
| | 2nd stage | 22:12 | |
| | 3rd stage | 30:11 | |
| | 4th stage | 35:12 | |
| | 5th stage | 34:1 | |
| | resultant ratio | 6545/132 = 49.5833 | 6545 turns of motors(or encoders) will make 132 turns of wheels |
| **Conversions** | ticks to metres | 0.000085292090497737556558 m/tick | |
| | ticks to radians | 0.002436916871363930187454 rad/tick | |
| | metres to ticks | 11724.4165802985662475159 1 tick/m | |
| | radian to ticks | 11.72441658029856624751591 tick/mm | |

CHAPTER 9

# Serial Protocol

## 9.1 About

A software program communicates with the robot by using predefined protocol on the serial or usb-serial lines. The provided c++ library, libkobuki.so does this for you, so in most cases, understanding the serial protocol is not necessary. This section is for implementers of libraries attempting to communciate with the Kobuki via either a different language (e.g. java) or their own custom c++ implementation.

In general, commands are sent to the robot on the RX line and responses / sensor readings are streamed back on the TX line at a rate of 20ms.

## 9.2 Data Types

### 9.2.1 Types

Data fields used in commands or payloads can be in the form of one of the three data types specified below:

| Name \| Description | | Bytes | Bits | Range | C/C++ Identifier | |
|---|---|---|---|---|---|---|
| Unsigned Byte | 8-bit unsigned int | 1 | 8 | 0~255 | unsigned char | uint8_t |
| Unsigned Short | 16-bit unsigned int | 2 | 16 | 0~65,535 | unsigned short | uint16_t |
| Unsigned Int | 32-bit unsigned int | 4 | 32 | 0~4,294,967,295 | unsigned int | uint32_t |

### 9.2.2 Ordering

Data for the multi-byte types are in LSB order. This means the least significant byte will come first in the bytestream, for example, the integer 2,864,434,397 (0xAABBCCDD) will be represented in the bytestream as:

| 0xDD | 0xCC | 0xBB | 0xAA |
|---|---|---|---|

## 9.3 The ByteStream

### 9.3.1 Structure

The returning stream consists of packets that combine both sensor data and responses to requests that have been sent in the previous cycle. A bytestream can be divided into 4 fields: Headers, Length, Payload and Checksum.

| **Headers** | | **Length** | **Payload** | | | **Checksum** |
|---|---|---|---|---|---|---|
| Header0 | Header1 | | SubPayload0 | ... | SubPayloadN-1 | |

| Name | Header 0 | Header 1 | Length | Payload | Checksum |
|---|---|---|---|---|---|
| **Size** | 1 Byte | 1 Byte | 1 Byte | N Bytes | 1 Byte |
| **Description** | 0xAA | 0x55 | Payload size | See below | XOR (length + payload) |

### 9.3.2 Headers

Two bytes of headers, header 0 and header 1, are of fixed value for both bytestreams, commands and feedback data. This headers are used to detect the starting point of bytestream.

### 9.3.3 Length

Length is a single byte that indicates size of the variable payload (in bytes). Length can be used to distinguish each bytestreams. Minimum value of this field is 3.

### 9.3.4 Payload

The payload is where the gold (actual data) is!

A **payload** is actually representative of several sub-payloads stitched together.

| Payload | | | | |
|---|---|---|---|---|
| SubPayload0 | SubPayload1 | SubPayload2 | ... | SubPayload N-1 |

**Sub-payloads** can be divided into three parts; Header, Length and Data:

| **Name** | Header | Length | Data |
|---|---|---|---|
| **Size** | 1 Byte | 1 Byte | N Byte(s) |
| **Description** | Identifier | Size of data | See below |

### 9.3.5 Checksum

The checksum is the XOR'ed value of the entire bytestream sans the headers. This is used as a check to ensure the integrity of the contents of the bytestream since individual bytes can be easily corrupted on the wire.

A c++ code snippet demonstrating the algorithm used:

```
unsigned int packet_size(buffer.size());
unsigned char cs(0);
for (unsigned int i = 2; i < packet_size; i++)
{
  cs ^= buffer[i];
}
return cs ? false : true;
```

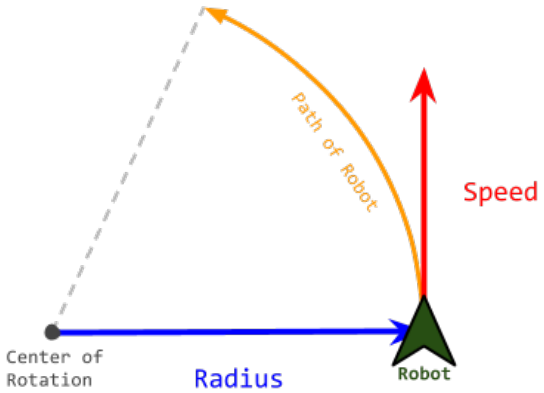
## 9.4 Command Packets

### 9.4.1 Command Identifiers

| ID | Name | Description |
|---|---|---|
| 1 | Base Control | Control wheel motors |
| 2 | Reserved | |
| 3 | Sound | Play custom sounds |
| 4 | Sound Sequence | Play predefined sound sequences |
| 5 | Reserved | |
| 6 | Reserved | |
| 7 | Reserved | |
| 8 | Reserved | |
| 9 | Request Extra | Request extra information |
| 10 | Reserved | |
| 11 | Reserved | |
| 12 | General Purpose Output | Control general purpose outputs |
| 13 | Set Controller Gain | Set PID gain of wheel velocity controller |
| 14 | Get Controller Gain | Get PID gain of wheel velocity controller |

### 9.4.2 Base Control

Control wheel motors to moving robot. Robot will follow the arc line, which radius is <Radius> mm, with <Speed> mm/s. Positive Radius indicates center of arc line that robot follows is located at the left side of the robot. Negative is opposite.




But actual value of speed field is little bit different. Here is conversion table.

| Motion | Speed(mm/s) | Radius(mm) |
|---|---|---|
| Pure Translation | Speed | 0 |
| Pure Rotation | w*b / 2 | 1 |
| Translation + Rotation | Speed * (Radius + b) / 2 ) / Radius, if Radius > 1 | Radius |
| | Speed * (Radius - b / 2 ) / Radius, if Radius < -1 | Radius |

- w is rotation speed of the robot, in [rad/s].
- b is bias or wheelbase, that indicates the length between the center of the wheels.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 1 | 0x01 | Fixed |
| **Length** | Size of data field | 1 | 4 | 0x04 | Fixed |
| **Data** | Speed | 2 | | | in mm/s |
| | Radius | 2 | | | in mm |

### 9.4.3 Sound

Play custom sounds with note and duration.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 3 | 0x03 | Fixed |
| **Length** | Size of data field | 1 | 3 | 0x03 | Fixed |
| **Data** | Note | 2 | | | 1 / (f*a), where f is the frequency (Hz), a is 0.00000275 |
| | Duration | 1 | | | Duration of playing note in milli-seconds |

**Note:** This command is implemented on the kobuki with firmware, but not implemented yet in the c++ library (kobuki_core).

### 9.4.4 Sound Sequence

Play predefined sounds by its index.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 4 | 0x04 | Fixed |
| **Length** | Size of data field | 1 | 1 | 0x01 | Fixed |
| **Data** | Sequence number | 1 | | | 0 for ON sound |
| | | | | | 1 for OFF sound |
| | | | | | 2 for RECHARGE sound |
| | | | | | 3 for BUTTON sound |
| | | | | | 4 for ERROR sound |
| | | | | | 5 for CLEANINGSTART sound |
| | | | | | 6 for CLEANINGEND sound |

### 9.4.5 Request Extra

Request extra data from robot. Especially version info of kobuki; Hardware Version, Firmware Version and Unique Device IDentifier(UDID)

UDID is unique to device. so can be used to identify on multiple robots.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 9 | 0x09 | Fixed |
| **Length** | Size of data field | 1 | 2 | 0x02 | Fixed |
| **Data** | Request flags | 2 | | | Set the flags to request extra data |
| | | | | | 0x01 for Hardware Version |
| | | | | | 0x02 for Firmware Version |
| | | | | | 0x08 for Unique Device ID |

### 9.4.6 General Purpose Output

This command has multiple roles. It controls LEDs, digital outputs and external powers.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 12 | 0x0C | Fixed |
| **Length** | Size of data field | 1 | 2 | 0x02 | Fixed |
| **Data** | Digital output flags 2 | | | | Set the flags to set high on output pins of expansion port |
| | | | | | 0x0001 for digital output ch. 0 |
| | | | | | 0x0002 for digital output ch. 1 |
| | | | | | 0x0004 for digital output ch. 2 |
| | | | | | 0x0008 for digital output ch. 3 |
| | | | | | |
| | | | | | Set the flags to turn on external powers |
| | | | | | 0x0010 for external power 3.3V ch. |
| | | | | | 0x0020 for external power 5V ch. |
| | | | | | 0x0040 for external power 12V/5A ch. |
| | | | | | 0x0080 for external power 12V/1.5A ch. |
| | | | | | |
| | | | | | Set the flags to turn on LEDs |
| | | | | | 0x0100 for red colour of LED1 |
| | | | | | 0x0200 for green colour of LED1 |
| | | | | | 0x0400 for red colour of LED2 |
| | | | | | 0x0800 for green colour of LED2 |

### 9.4.7 Set Controller Gain

Set PID gain of wheel velocity controller of robot.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 1 | 0x01 | Fixed |
| **Length** | Size of data field | 1 | 13 | 0x0D | Fixed |
| **Data** | Type | 1 | | | 0 for factory-default PID gain |
| | | | | | 1 for user-configured PID gain |
| | P gain | 4 | | | Kp * 1000 (default: 100*1000) |
| | I gain | 4 | | | Ki * 1000 (default: 0.1*1000) |
| | D gain | 4 | | | Kd * 1000 (default: 2*1000) |

### 9.4.8 Get Controller Gain

Request PID gain of wheel velocity controller of robot.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 1 | 0x01 | Fixed |
| **Length** | Size of data field | 1 | 14 | 0x0E | Fixed |
| **Data** | unused | 1 | | | |

## 9.5 Feedback Packets

### 9.5.1 Feedback Identifiers

| ID | Name | Descritpion | Availability |
|---|---|---|---|
| 1 | Basic Sensor Data | Basic core sensor data | By default |
| 2 | Reserved | | |
| 3 | Docking IR | Signals from docking station | By default |
| 4 | Inertial Sensor | Gyro data both angle and angular velocity | By default |
| 5 | Cliff | PSD data facing floor | By default |
| 6 | Current | Current of wheel motors | By default |
| 7 | Reserved | | |
| 8 | Reserved | | |
| 9 | Reserved | | |
| 10 | Hardware Version | Version number of kobuki hardware | On request |
| 11 | Firmware Version | Version number of kobuki firmware | On request |
| 12 | Reserved | | |
| 13 | Raw data of 3-axis gyro | Raw ADC data of digital 3-axis gyro | By default |
| 14 | Reserved | | |
| 15 | Reserved | | |
| 16 | General Purpose Input | Inputs from 25-pin expansion port | By default |
| 17 | Reserved | | |
| 18 | Reserved | | |
| 19 | Unique Device IDentifier(UDID) | Unique number to identify robot | On request |
| 20 | Reserved | | |
| 21 | Controller Info | PID gain values of wheel velocity controller | On request |

### 9.5.2 Basic Sensor Data

**Note:** This sub-payload is always streamed.

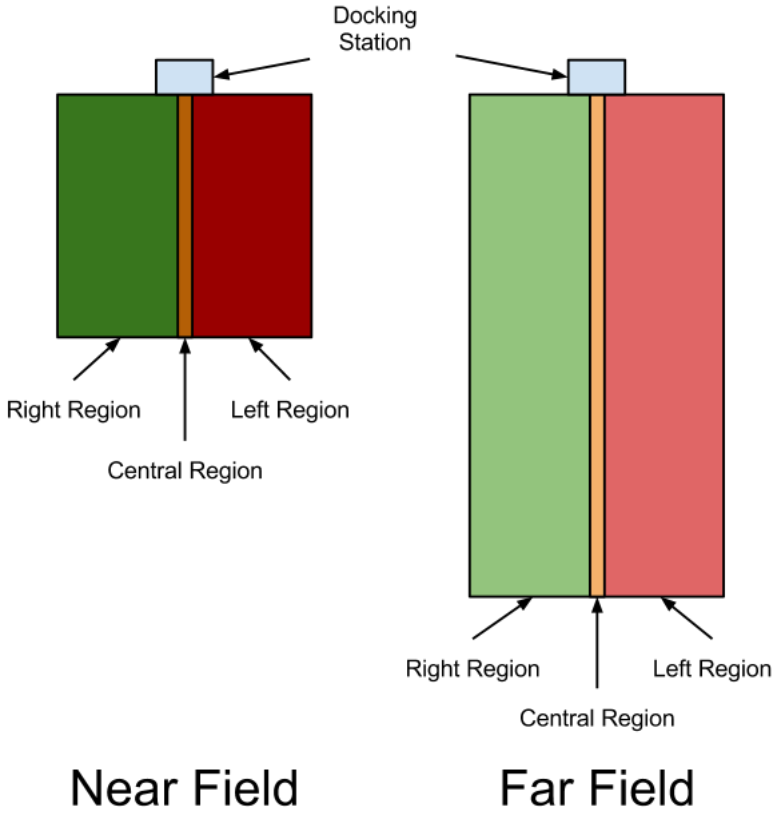
| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Feedback Identifier | 1 | 1 | 0x01 | Fixed |
| **Length** | Size of data field | 1 | 15 | 0x0F | Fixed |
| **Data** | Timestamp | 2 | | | Timestamp generated internally in milliseconds |
| | | | | | It circulates from 0 to 65535 |
| | Bumper | 1 | | | Flag will be setted when bumper is pressed |
| | | | | | 0x01 for right bumper |
| | | | | | 0x02 for central bumper |
| | | | | | 0x04 for left bumper |
| | Wheel drop | 1 | | | Flag will be setted when wheel is dropped |
| | | | | | 0x01 for right wheel |
| | | | | | 0x02 for left wheel |
| | Cliff | 1 | | | Flag will be setted when cliff is detected |
| | | | | | 0x01 for right cliff sensor |
| | | | | | 0x02 for central cliff sensor |
| | | | | | 0x04 for left cliff sensor |
| | Left encoder | 2 | | | Accumulated encoder data of left and right wheels in ticks |
| | | | | | Increments of this value means forward direction |
| | | | | | It circulates from 0 to 65535 |
| | Right encoder | 2 | | | As above |
| | Left PWM | 1 | | | PWM value that applied to left and right wheel motor |
| | | | | | This data should be converted signed type to represent correctly |
| | | | | | Negative sign indicates backward direction |
| | Right PWM | 1 | | | As above |
| | Button | 1 | | | Flag will be setted when button is pressed |
| | | | | | 0x01 for Button 0 |
| | | | | | 0x02 for Button 1 |
| | | | | | 0x04 for Button 2 |
| | Charger | 1 | | | 0 for DISCHARGING state |
| | | | | | 2 for DOCKING_CHARGED state |
| | | | | | 6 for DOCKING_CHARGING state |
| | | | | | 18 for ADAPTER_CHARGED state |
| | | | | | 22 for ADAPTER_CHARGING state |
| | Battery | 1 | | | Voltage of battery in 0.1 V |
| | | | | | Typically 16.7 V when fully charged |
| | Overcurrent flags | 1 | | | Flag will be setted when overcurrent is detected |
| | | | | | 0x01 for left wheel |
| | | | | | 0x02 for right wheel |

### 9.5.3 Docking IR

Signals from the docking station.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 3 | 0x03 | Fixed |
| **Length** | Size of data field | 1 | 3 | 0x03 | Fixed |
| **Data** | Right signal | 1 | | | Flag will be setted when signal is detected |
| | | | | | 0x01 for NEAR_LEFT state |
| | | | | | 0x02 for NEAR_CENTER state |
| | | | | | 0x04 for NEAR_RIGHT state |
| | | | | | 0x08 for FAR_CENTER state |
| | | | | | ox10 for FAR_LEFT state |
| | | | | | 0x20 for FAR_RIGHT state |
| | Central signal | 1 | | | |
| | Left signal | 1 | | | |

Kobuki's docking station has 3 IR emitters. The emitted IR lights cover three regions in front of the docking station: left, central and right, each divided in two sub-fields: near and far. Each beam encodes this information, so the robot knows at any moment in which region and sub-field he is. Also, as regions and fields are independently identified, they can be overlap on its borders.




### 9.5.4 Inertial Sensor Data

**Note:** This sub-payload is always streamed.

Z-axis gyro data only available.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 4 | 0x04 | Fixed |
| **Length** | Size of data field | 1 | 7 | 0x07 | Fixed |
| **Data** | Angle | 2 | | | Factory calibrated |
| | Angle rate | 2 | | | Factory calibrated |
| | Unused | 1 | | | |
| | Unused | 1 | | | |
| | Unused | 1 | | | |

### 9.5.5 Cliff Sensor Data

**Note:** This sub-payload is always streamed.

This value is related with distance between sensor and floor surface. See the datasheet for more detailed information.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 5 | 0x05 | Fixed |
| **Length** | Size of data field | 1 | 6 | 0x06 Fixed | |
| **Data** | Right cliff sensor | 2 | | | ADC output of each PSD |
| | | | | | Data range: 0 ~ 4095 (0 ~ 3.3V) |
| | | | | | Distance range: 2 ~ 15 cm |
| | | | | | Distance is not linear w.r.t. ADC output. |
| | | | | | See the datasheet for more detail. |
| | Central cliff sensor | 2 | | | As above |
| | Left cliff sensor | 2 | | | As above |

### 9.5.6 Current

**Note:** This sub-payload is always streamed.

Current sensor readings of wheel motors.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 6 | 0x06 | Fixed |
| **Length** | Size of data field | 1 | 2 | 0x02 | Fixed |
| **Data** | Left motor | 2 | | | in 10mA |
| | Right motor | 2 | | | in 10mA |

### 9.5.7 Hardware Version

**Note:** This sub-payload is sent only on request.

Hardware version info in triplet form; <major>.<minor>.<patch>

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 10 | 0x0A | Fixed |
| **Length** | Size of data field | 1 | 4 | 0x04 | Fixed |
| **Data** | Patch | 1 | | | |
| | Minor 1 | | | | |
| | Major 1 | | | | |
| | Unused | 1 | 0 | 0x00 | Fixed |

### 9.5.8 Firmware Version

**Note:** This sub-payload is sent only on request.

Firmware version info in triplet form; <major>.<minor>.<patch>

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 11 | 0x0A | Fixed |
| **Length** | Size of data field | 1 | 4 | 0x04 | Fixed |
| **Data** | Patch | 1 | | | |
| | Minor 1 | | | | |
| | Major 1 | | | | |
| | Unused | 1 | 0 | 0x00 | Fixed |

### 9.5.9 Raw Data Of 3D Gyro

**Note:** This sub-payload is always streamed.

Raw ADC data of digital 3D gyro L3G4200D. Due to difference of acquisition rate and update rate, 2-3 data will be arrived at once. Digit to deg/s ratio is 0.00875, it comes from datasheet of 3d gyro.

ADC output of each-axis is in 0.00875 deg/s.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 13 | 0x0D | Fixed |
| **Length** | Size of data field | 1 | 2+6N | | |
| **Data** | Frame id | 1 | | | Frame id of 'Raw gyro data 0' |
| | | | | | Every sensor readings can identified by frame id |
| | | | | | Circulates from 0 to 255 |
| | Followed data length | 1 | 3N | | |
| | Raw gyro data 0 | 2 | | | x-axis |
| | | 2 | | | y-axis |
| | | 2 | | | z-axis |
| | ... | 2 | | | z-axis |
| | Raw gyro data N-1 | 2 | | | |
| | | 2 | | | |
| | | 2 | | | |

**Note:** Sensing axis of 3d gyro is not match with robot. It is rotated 90 degree counterclockwise about z-axis. So, below conversion will needed.

```
const double digit_to_dps = 0.00875;
angular_velocity.x =  -digit_to_dps * (short)raw_gyro_data.y;
angular_velocity.y =   digit_to_dps * (short)raw_gyro_data.x;
angular_velocity.z =   digit_to_dps * (short)raw_gyro_data.z;
```

### 9.5.10 General Purpose Input

**Note:** This sub-payload is always streamed.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 16 | 0x10 | Fixed |
| **Length** | Size of data field | 1 | 16 | 0x10 | Fixed |
| **Data** | Digital input | 2 | | | Flag will be setted, when high voltage is applied |
| | | | | | 0x01 for digital input ch. 0 |
| | | | | | 0x02 for digital input ch. 1 |
| | | | | | 0x04 for digital input ch. 2 |
| | | | | | 0x08 for input output ch. 3 |
| | Analog input ch.0 | 2 | | | 12-bit ADC output of each channel |
| | | | | | Data range: 0 ~ 4095(2^12-1) |
| | | | | | Voltage range: 0 ~ 3.3 V |
| | Analog input ch.1 | 2 | | | As above |
| | Analog input ch.2 | 2 | | | As above |
| | Analog input ch.3 | 2 | | | As above |
| | Unused | 2 | | | |
| | Unused | 2 | | | |
| | Unused | 2 | | | |

### 9.5.11 Unique Device IDentifier (UDID)

**Note:** This sub-payload is sent only on request.

Contains Unique Device IDentifier of robot. This value is unique for all robot in the world. It can be represented by triplet form: <UDID0>-<UDID1>-<UDID2>

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 19 | 0x13 | Fixed |
| **Length** | Size of data field | 1 | 12 | 0x0C | Fixed |
| **Data** | UDID0 | 4 | | | |
| | UDID1 | 4 | | | |
| | UDID2 | 4 | | | |

### 9.5.12 Controller Info

**Note:** This sub-payload is sent only on request.

Contains PID gain of wheel velocity controller of robot.

| | Name | Size | Value | Hex | Description |
|---|---|---|---|---|---|
| **Header** | Identifier | 1 | 1 | 0x01 | Fixed |
| **Length** | Size of data field | 1 | 21 | 0x15 | Fixed |
| **Data** | Type | 1 | | | Current controller setup |
| | | | | | 0 for factory-default PID gain |
| | | | | | 1 for user-configured PID gain |
| | P gain | 4 | | | Kp * 1000 (default: 100*1000) |
| | I gain | 4 | | | Ki * 1000 (default: 0.1*1000) |
| | D gain | 4 | | | Kd * 1000 (default: 2*1000) |

CHAPTER 10

# Firmware

## 10.1 Versioning

Firmware versions follow semantic versioning rules. The c++ driver checks for compatibility between the software (i.e. driver) and firmware. Firmware versions are of the form M.m.p:

- **M(ajor)** versions typically break protocol compatibility. When software and firmware are incompatible, the software will emit an error, suggest the required update and shutdown.
- **m(inor)** versions add features, but the protocol will have not been modified. Software and firmware will interoperate, but warnings will be issued just-in-time when features are used that aren't supported by the connected firmware.
- **p(atch)** versions provide minor bugfixes, but do not break driver or protocol compatibility.

Additionally, the software maintains a list of recommended versions. Even if there is only a minor or patch version difference, it will give you a warning on connection and suggest the recommended firmware version to upgrade to. For example:

```
$ kobuki-simple-keyop

Simple Keyop : Utility for driving kobuki by keyboard.

Reading from keyboard
---------------------
Forward/back arrows : linear velocity incr/decr.
Right/left arrows : angular velocity incr/decr.
Spacebar : reset linear/angular velocities.
q : quit.

[WARNING] The firmware does not match any of the recommended versions for this␣
↪software.
[WARNING] Consider replacing the firmware. For more information,
[WARNING] refer to https://kobuki.readthedocs.io/en/devel/firmware.html.
[WARNING]  - Firmware Version: 1.1.3
```

(continues on next page)

(continued from previous page)

```
[WARNING]  - Recommended Versions: 1.1.4 / 1.2.0

current pose: [x: 5.61871e-310, y: 1.57358e-314, heading: 6.90938e-310]
current pose: [x: 5.61871e-310, y: 1.57358e-314, heading: 6.90938e-310]
```

The c++ driver provides a utility for checking the version that is running on your kobuki. It will also provide versioning information for the driver (software) and hardware:

```
$ kobuki-version-info
  Version Info:
    * Hardware Version: 1.0.4
    * Firmware Version: 1.2.0
    * Software Version: 1.0.0
    * Unique Device ID: 97713968-842422349-1361404194
```

Additionally, firmware binaries come in three flavours:

- **latest**: most recent, but be aware that this version hasn't been tested much
- **stable**: more recent than factory and reasonably well tested
- **factory**: flashed onto the robots at the factory, has undergone stress testing

These are identified by the trailing suffix on binary filenames stored in the kobuki_firmware repository. More details on the specific features / fixes provided by each version can be found in the kobuki firmware CHANGELOG.

## 10.2 Updating Firmware

Kobuki's come pre-flashed from the factory. The only time you should need to upgrade is if you have an older version and wish to catch new fixes or features.

### 10.2.1 Linux

#### The Flashing Utility

```
# Download stm32flash-0.4.gz from https://sourceforge.net/projects/stm32flash/files/
$ tar -xvzf stm32flash-0.4.tar.gz
$ cd stm32flash
$ make
```

#### Download Firmware

```
# Choose & download from https://github.com/kobuki-base/kobuki_firmware/tree/devel/
↪firmware
# e.g. latest
$ wget --no-check-certificate --content-disposition https://github.com/kobuki-base/
↪kobuki_firmware/blob/devel/firmware/kobuki_firmware_1.2.0-latest.hex?raw=true
```

### Identify The COM Port

If you have a udev rule installed, it will show up as */dev/kobuki*. If not, you can typically find it under one of the ttyUSB ports, e.g. /dev/ttyUSB0. If you are not sure, type *dmesg* into a terminal, unplug and replug the robot and type *dmesg* again. You should now be able to see which port is assigned to the robot.

### Switch to Download Mode

1. Connect the robot to your PC using the USB cable
2. Turn off the robot (switch on the side)
3. Switch from normal runtime mode to firmware download mode

This simply changes the type of data that is sent back and forth along the usb connection. You can do this by moving the switch illustrated below into the 'download' (up) position. Note that this switch is embedded into the robot cover so it isn't easily thrown by accident - you may need thin plyers or some similar tool. You can find the mode switch mechanism on the right side of the control panel:




### Flashing

---

**Note:** The following instructions assume flashing of kobuki_firmware_1.2.0-latest.hex and port /dev/ttyUSB0. Modify these as necessary.

---

**Warning:** you need to execute the flashing command IMMEDIATELY after turning the robot on!

1. Turn off the robot
2. Check that the switch is in download mode
3. Turn on the robot

```
$ ./stm32flash -b 115200 -w kobuki_firmware_1.2.0-latest.hex /dev/ttyUSB0
  stm32flash 0.4

  http://stm32flash.googlecode.com/
```

(continues on next page)

(continued from previous page)

```
Using Parser : Intel HEX
Interface serial_posix: 115200 8E1
Version      : 0x22
Option 1     : 0x00
Option 2     : 0x00
Device ID    : 0x0414 (High-density)
- RAM        : 64KiB  (512b reserved by bootloader)
- Flash      : 512KiB (sector size: 2x2048)
- Option RAM : 16b
- System RAM : 2KiB
Write to memory
Erasing memory
Wrote address 0x0800a3f0 (100.00%) Done.
```

#### Reboot

- Turn off the robot power
- Flick the firmware switch back to 'Operation' mode.
- Turn on the robot power
- I'm happy, you should be too!

### 10.2.2 Windows

#### The Flashing Utility

- Find, download and install *Flash_Loader_Demonstrator_v2.5.0_Setup.exe*.

#### Download Firmware

Choose & download from kobuki_firmware/firmware.

#### Identify the COM Port

Usually this will show up on COM1, but check to make sure.

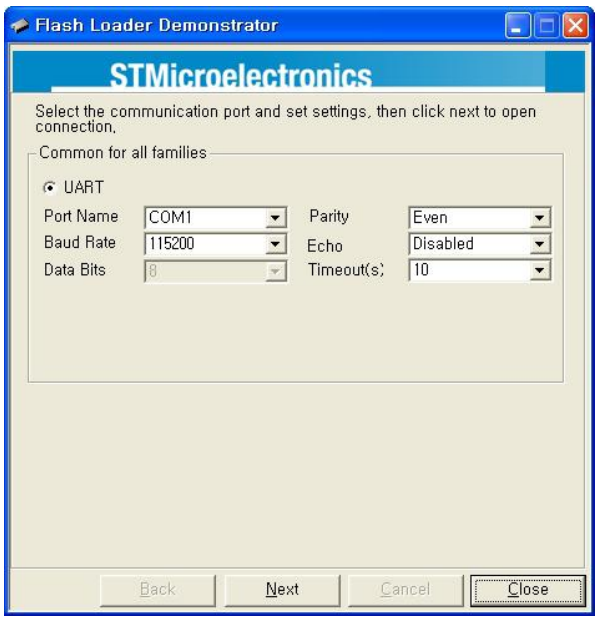
#### Switch to Download Mode

1. Connect the robot to your PC using the USB cable
2. Turn off the robot (switch on the side)
3. Switch from normal runtime mode to firmware download mode

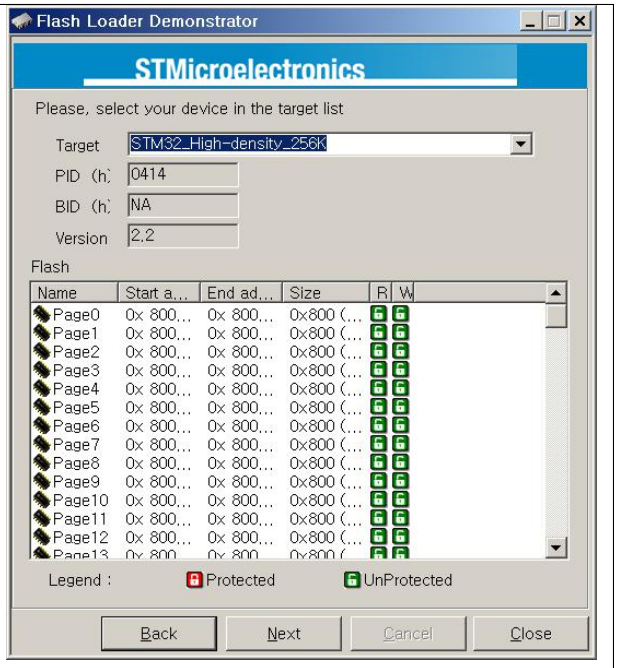
This simply changes the type of data that is sent back and forth along the usb connection. You can do this by moving the switch illustrated below into the 'download' (up) position. Note that this switch is embedded into the robot cover so it isn't easily thrown by accident - you may need thin plyers or some similar tool. You can find the mode switch mechanism on the right side of the control panel - see the image below.

### Flashing

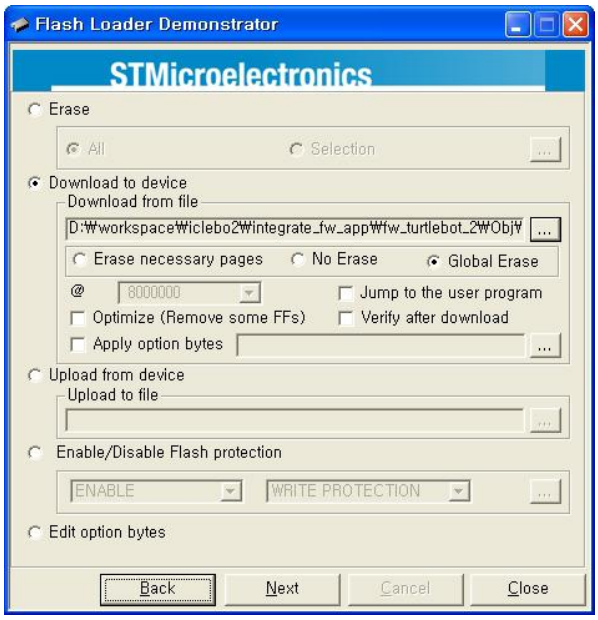
1. Turn off the robot
2. Check that the switch is in download mode
3. Turn on the robot



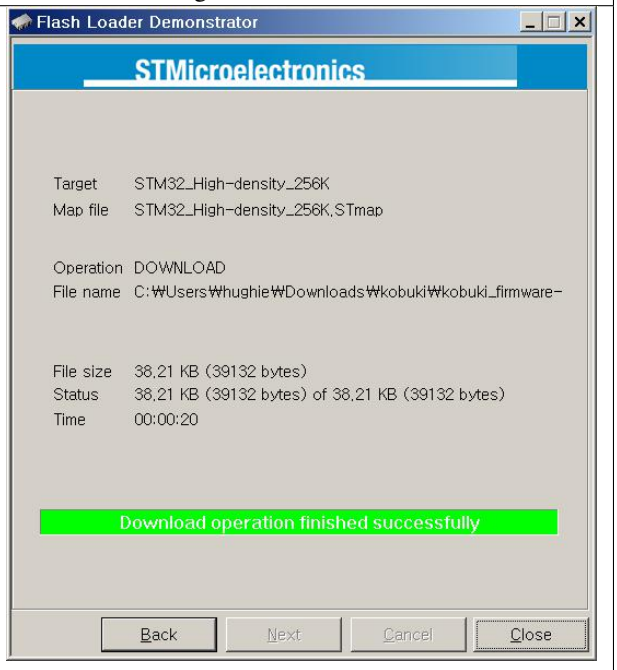

Configure Properties




Check that the target is identified




Enter the Download from file (your .hex)




Success!

#### Rebooting

- Turn off the robot power
- Flick the firmware switch back to 'Operation' mode.
- Turn on the robot power
- I'm happy, you should be too!

## 10.3 Special Firmware Modes

### 10.3.1 Activating

Kobuki has some special firmware modes, which can be activated on startup.

- Random Walker
- Arduino/Embedded Board support mode

To activate one of them, follow these instructions:

- Turn on Kobuki.
- Within in the first 3 seconds press and hold either button BO (Random Walker) or B1 (Arduino) for 2 seconds
- If you see LED2 (Random Walker) or LED1 (Arduino) switching between red and green, your chosen mode has been activated.

---

**Note:** These modes have been introduced to the firmware with version 1.1.0. In case your Kobuki is not running this or a later version, please refer to the section about updating the firmware.

---

### 10.3.2 Random Walker Mode

In random walker mode Kobuki is driving around until it hits an object with the bumper or a cliff is detected. In both cases, Kobuki will stop, turn by a random amount of degrees and continue driving .

**Warning:** In this mode Kobuki's wheel drop sensors are not activated. So, be careful when lifting up Kobuki!

### 10.3.3 Arduino / Embedded Board Support Mode

In this mode the serial port (DB25 connector) gives access to basic controls of Kobuki. You can hook up the digital/analog inputs/outpus of your Arduino or other embedded boards and start writing simple control programs.

Below is the special pin setting listed. Please refer to the serial port description for the name to pin mapping.

- DI0: Not used
- DI1: Not used
- DI2: Not used
- DI3: Not used
- DO0: Bumper left (pressed/released)

- DO1: Bumper centre (pressed/released)
- DO2: Bumper right (pressed/released)
- DO3: Wheel drop sensors (at least one wheel is dropped / none is dropped)
- AI0: Wheel speed right (0V - full speed backward, 3.3V - full speed forward)
- AI1: Wheel speed left (0V - full speed backward, 3.3V - full speed forward)
- AI2: Not used
- AI3: Not used

All other pins (GND, RX, TX etc.) remain unchanged.

---

**Note:** To enable the motors you need to press button B0.

---

CHAPTER 11

# Media


- Kobuki Images & Renderings
- Marketing Materials

CHAPTER 12

# Docking Stations

## 12.1 About

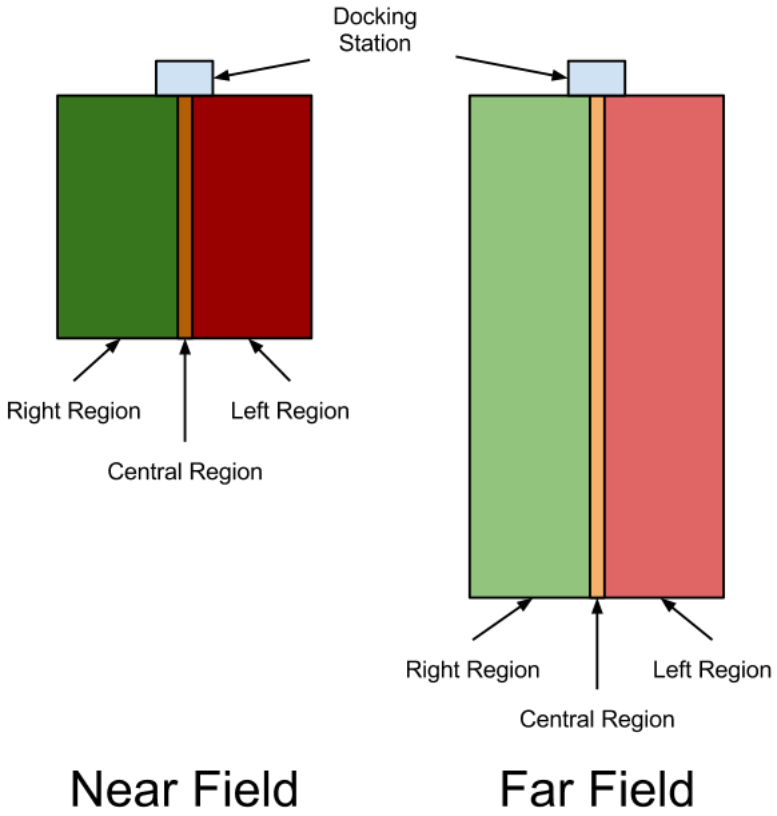


Docking stations are an optional extra that can enable Kobuki to autonomously recharge as needed (with a little programming).

## 12.2 How it Works

Kobuki's docking station has 3 IR **emitters** positioned on the left, right and centre of the docking station. Each emitter beams encoded signals in a manner that will ensure coverage of the area in front of the docking station partitioned in

six areas - left, right and centre, further subdivided near and far as illustrated below.




Three **receivers** are also positioned left, right and centre on the docking station and will receive a mix of the signals hitherto sent from the emitters and bounced off any object in range of the docking station (i.e. an incoming Kobuki). If for example, Kobuki was still far from the docking station and in the left region, then both left and centre signals will receive the *FAR_LEFT* state and the right receiver will record null. This is sufficient to enable a simple docking algorithm to work robustly, even if the solution may not always be elegant due to a lack of resolution on the range axis (merely bifurcates, near and far).

## 12.3 Software

**Todo:** In need of someone owning a docking station to assist with usage of kobuki_dock_drive library and example demo

# CHAPTER 13

# Embedded Boards

## 13.1 Cross Compiling

### 13.1.1 Getting Started

Kobuki is c++ and built using CMake, so to cross-compile, these instructions will take advantage of CMake Toolchains, configuration which of is dependent on the c++ toolchain being used to compile the libraries.

For a primer on CMake and how to define CMake toolchains, refer to the cmake manual - cmake toolchains.

### 13.1.2 Use Case - arm-linux-gnueabihf

#### Preparation

Let's get hands on and use one of the c++ toolchains provided by Ubuntu 20.04 as an example.

**Note:** You are not limited by what your linux distro provides, pretty much any downloadable gcc toolchain can be enabled this one, just merely point your cmake toolchain configuration to wherever you have installed your toolchain.

Download a toolchain:

```
sudo apt install g++-arm-linux-gnueabihf
```

This is the generic toolchain for arm cores with hard-float capabilities (usually the more powerful variety of arm cores). You will find the toolchain installed in /usr/arm-linux-gnueabihf/. Next, create a cmake toolchain file that will instruct cmake on where to find your toolchain, your your staging area and set appropriate CXX Flags for your target:

```
set(TOOLCHAIN_TUPLE "arm-linux-gnueabihf" CACHE STRING "Toolchain signature␣
↪identifying cpu-vendor-platform-clibrary.")
set(TOOLCHAIN_ROOT "/usr/${TOOLCHAIN_TUPLE}" CACHE STRING "Root of the target␣
↪development environment (libraries, headers etc).")
```

(continues on next page)

(continued from previous page)

```
# Target information
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR "arm")
unset(CMAKE_Fortran_COMPILER)  # This toolchain doesn't have a fortran compiler
set(CMAKE_C_COMPILER   ${TOOLCHAIN_TUPLE}-gcc) # Make sure these are in your PATH
set(CMAKE_CXX_COMPILER ${TOOLCHAIN_TUPLE}-g++)

# Search paths - only dig around in the toolchain root and staging area
set(CMAKE_FIND_ROOT_PATH "${TOOLCHAIN_SYSROOT};${CMAKE_CURRENT_LIST_DIR}/install"␣
↪CACHE STRING "Cmake search variable for finding libraries/headers.")
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER) # Don't search for programs outside of␣
↪CMAKE_FIND_ROOT_PATH and CMAKE_SYSROOT
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)  # ... libraries
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)  # ... headers
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)  # ... cmake modules

# CXX Flags specific to the target platform (typical raspberry pi platform)
#
#  - benchmark yourself, mileage will vary considerably, large speedups to be gained
#  - a good starting point is https://wiki.gentoo.org/wiki/Safe_CFLAGS#ARMv6.
↪2FARM1176JZF-S
#
# Also, -Wno-psabi avoids irritating gcc 7.1 warnings about not mixing binaries with␣
↪gcc 6 binaries
#
set(CMAKE_CXX_FLAGS "-march=armv7 -mtune=arm1176jzf-s -pipe -mfloat-abi=hard -
↪mfpu=vfp -Wno-psabi" CACHE STRING "flags specific for an armv7/arm1176jzf-s platform
↪")

# Hide from cache's front page
MARK_AS_ADVANCED(CMAKE_GENERATOR CMAKE_FIND_ROOT_PATH CMAKE_TOOLCHAIN_FILE TOOLCHAIN_
↪FAMILY TOOLCHAIN_TUPLE)
```

It can be named whatever you please, here we'll refer to it as arm-linux-gnueabihf.cmake. In other circumstances, toolchain, staging area and cxx flags would be handled separately for maximum flexibility, but one file here keeps things simple to get started.

### Building

Follow the instructions for setting up the sources as in *Software - Preparation*, but stop short of building, we'll do that a little differently here. Namely:

1. Configure your PATH so that your toolchain can be found
2. Point cmake at your toolchain file

The modified instructions for building:

```
$ export PATH=${PATH}:/usr/arm-linux-gnueabihf/bin
$ export CMAKE_ARGS="-DBUILD_TESTING=OFF --no-warn-unused-cli"
$ export CROSS_COMPILE_ARGS=-DCMAKE_TOOLCHAIN_FILE=`pwd`/arm-linux-gnueabihf.cmake
$ colcon build --merge-install --cmake-args ${CMAKE_ARGS} ${CROSS_COMPILE_ARGS}
```

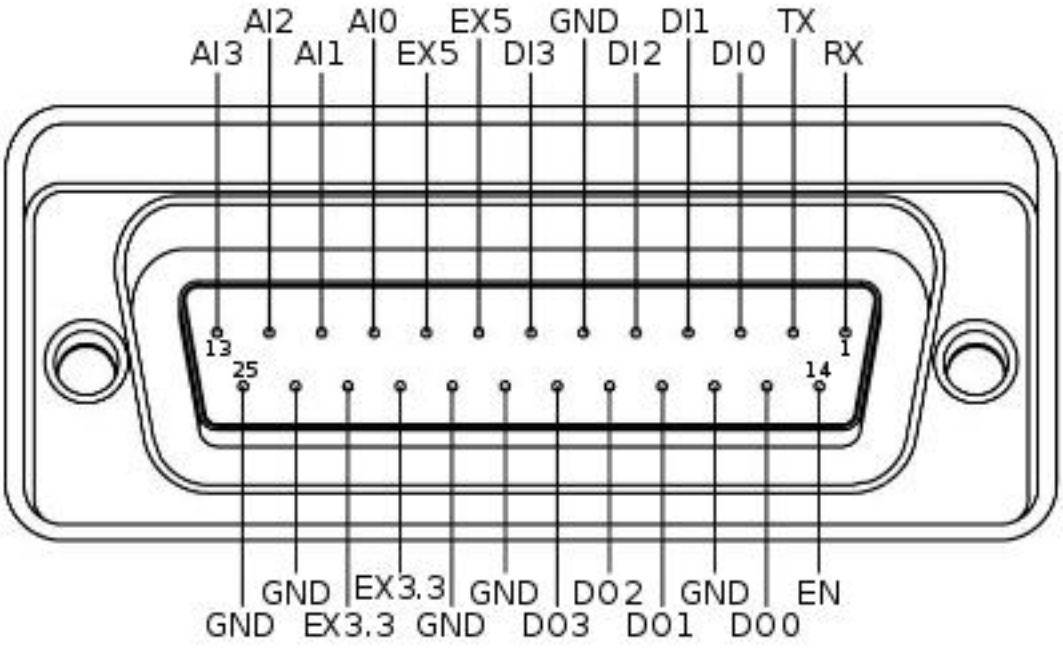
Other variations on the build step still hold as per the instructions in *Software - Build*.

These instructions are continuously vetted with a github action (yaml, results/logs).

## 13.2 Using The Serial Port (!USB)

If your embedded board has a serial port rather than a USB, you're in luck, Kobuki has that too via it's expansion port. You most likely will have to wire your own cable to make the correct pin-to-pin connections, as outlined in the section on the *Expansion Port*.

Reproducing here for convenience:




The minimum number of required pins for serial communication is three; TX, RX, and GND. Additionally EX3.3 or EX5 can be used for powering external devices, such as line transceiver.

Once connected, you should find your kobuki on one of the */dev/ttySN* ports (N = 1, 2, . . . ). Simply pass that string as the serial port identifier in the initialisation phase of your software applications.

CHAPTER 14

# Hardware Extensions

## 14.1 Overview

Kobuki provides additional *Power* connections and an *Expansion Port* with analog & digital io (and additional power connections) that let you extend the capabilities of your Kobuki in wierd and wonderful ways. Refer to those links for specific details.

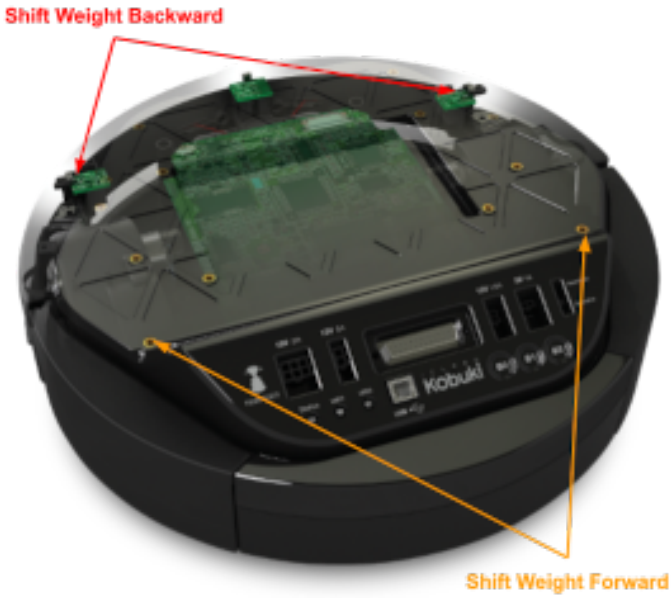
Some interesting examples to follow.

## 14.2 Use Case - Payload Balancing

**Note:** Kobuki's ancestry belongs to cleaning robot lines and consequently wasn't designed for handling variations in payload. Sometimes you'll need to provide some assistance!

When you start mounting additional equipment on Kobuki, the kinematic and dynamic motion envelopes will start to be considerably affected as the centre of gravity shifts. This is particularly important when it comes to Kobuki's ability to traverse small obstacles or slopes as it travels in the longitudinal direction.

In these cases, simply design some weights that can be mounted, like ballast, to shift the centre of gravity to where you need it. An example of such were the metal cylindrical pipes designed for the Turtlebot 2. These could be affixed around poles attached to the screws on the back of the kobuki.

| | |
|---|---|
|  |  |
| Steel Anchor Weights | Placement |

- Reference model diagrams for counterweights: igs, stp

## 14.3 Use Case - 3D Sensor

Depth sensors are typically connected to the 12V 1.5A power supply and via USB to your compute board (netbook/embedded board). To do this you'll need to modify the power capable with the connector specified in the *Power* section.

**Todo:** Could really use a step-by-step pictorial walkthrough.

## 14.4 Use Case - Laptop Recharging

**Note:** Recharging mode will only activate when the Kobuki itself is being recharged (otherwise the current draw would have a detrimental affect on runtime performance). This is true for both dock / cable recharging.

Similar to the depth sensor use case, you'll need to modify the recharging cable for a netbook and attach it to the 19V@2A connector (refer to the *Power* section) to take advantage of the on-board power supply. This is not only extremely convenient (no more detaching/reattaching) but will also permit you to completely automate your application.

Most turtlebot 2 suppliers would provide a netbook / modified cable with the full solution.

**Todo:** Could really use a step-by-step pictorial walkthrough.

## 14.5 Use Case - IR Sensor Array

Kobuki usually gets equipped with a 3d sensor, however, these typically have limitations for the purposes of obstacle avoidance:

- Narrow fov (58° x 43° horizontal x vertical)
- Death zone in the first 45 cm
- Cannot detect glass walls
- Cannot reliably detect polished metallic or very black surfaces

In one experiment an 11 IR sensors half ring, pointing 12 degrees downward was added to compensate.

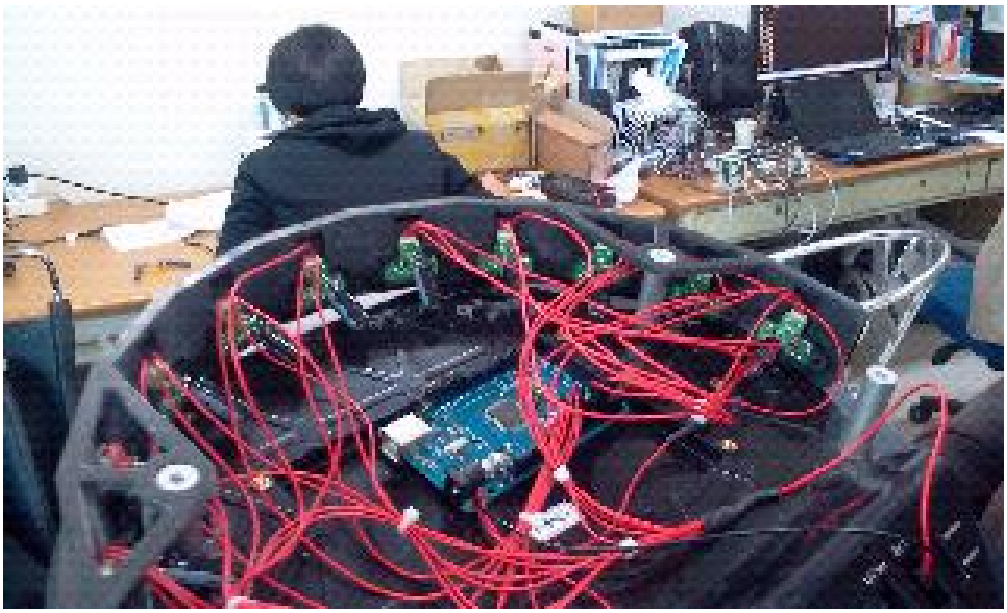
- Sensor model: Sharp GP2Y0A21YK
- Power supply: Kobuki's 5V, 1A
- Sensor reading: Arduino MEGA 2560
- PC interface: Arduino custom firmware – Bosch adc_driver
- Mounting: 3D printed frame.

| | | |
|---|---|---|
|  |  |  |
| PSD Array I | PSD Array II | Sonar Array |

The analog output of sensors is read by the Arduino board, while for power and ground they are connected to Kobuki's 5V 1A power source. Connecting several sensors to the same power supply makes readings very noisy when there aren't obstacles. The solution was to put decoupling capacitors on each sensor. For interfacing Arduino, we use Bosch adc_driver.

Different mounting frames are available for downloading and printing in our file server:

- Horizontally mounted MaxBotix's LV-Maxsonars
- Horizontally mounted Sharp IR sensors
- 12 degrees downward pointing Sharp IR sensors



CHAPTER 15

# Non-C++ Kobuki



The only requirement to programming with Kobuki in a language other than C++ is the ability to communicate with a serial port. To do so, you'll need to implement the *Serial Protocol* in the language of your choice. The c++ library can be a useful guide in how to do so.

To date there have been several experimental java/android implementations that have made this journey.

CHAPTER 16

# Documentation

The ROS1 documentation for Kobuki can be found on the ROS Wiki.

CHAPTER 17

# Installation

# CHAPTER 18

...

# CHAPTER 19

# Changelog

**Note:** This is a curated list of changes for all repositories in the kobuki ecosystem (to which this documentation pertains). See also:

[kobuki_core/CHANGELOG.rst] [kobuki_firmware/CHANGELOG.rst]

## 19.1 September '20

- **[kobuki_core-1.3.1]**
  - configurable stdout logging
  - custom_logging and raw_data_stream demos added
  - dual version firmware compatibility (1.1.4, 1.2.0)
- **[kobuki_core-1.3.0]** LegacyPose2D -> Eigen::Vector3d
- **[kobuki_core-0.7.10]** dual version firmware compatibility (1.1.4, 1.2.0)
- **[kobuki_documentation-1.0.2]** debugging tutorials (logging and raw data streams)

## 19.2 August '20

- **[kobuki_core-1.2.0]** kobuki_driver & kobuki_dock_driver merged into kobuki_core
- **[kobuki_core-1.1.1]** (bugfix) restore low latency usb reads via the udev rule and doxygen revamp
- **[kobuki_core-0.7.9]** (bugfix) restore low latency usb reads via the udev rule
- **[kobuki_documentation-1.0.1]** cross-compiling instructions
- **[kobuki_documentation-1.0.0]** new guide on readthedocs, everything in one place now!

## 19.3 Mar '20

- **[kobuki_firmware-1.2.0]** new github repo for the kobuki firmware binaries, with license

## 19.4 Jan '20

- **[kobuki_core-1.0.0]**
  - moved to the kobuki-base github org
  - ported to the colcon build system

# CHAPTER 20

# Glossary

**kobuki** kobuki: n. korean for turtle

**fsm**

**flying spaghetti monster** Whilst a serious religous entity in his own right (see pastafarianism), it's also very easy to imagine your code become a spiritual flying spaghetti monster if left unchecked:

```
  _  _(o)_(o)_  _
._\`:_ F S M _:' \_,
    / (`---'\ `-.
 ,-`  _)     (_,
```

# Index


## F

flying spaghetti monster, **85**
fsm, **85**

## K

kobuki, **85**