Getting Started

Overview

The **kobuki::Kobuki** class is the first port of call for developing your kobuki program. All kobuki configuration and interaction is done via this class. This guide will cover configuration and interaction with the library method api. Interactions via callbacks using sigslots is covered especially in the guide on **sigslots**.

Configuration

Kobuki configuration is handled by the **kobuki::Parameters** class which is passed to 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 typically the serial **device_port**. Note that on some systems you can configure a rule (refer to the **ftdi serial device guide**) for the device port, otherwise you will require one of the usual COM1 or /dev/ttyUSB0 values.

```
#include <iostream>
#include <kobuki_driver/kobuki.hpp>
#include <ecl/time.hpp>

class KobukiManager {
public:
  KobukiManager() {
    kobuki::Parameters parameters;
    // change the default device port from /dev/kobuki to /dev/ttyUSB0
    parameters.device_port = "/dev/ttyUSB0";
    // Other parameters are typically happy enough as defaults
    // namespaces all sigslot connection names under this value, only important if you want
to
    parameters.sigslots_namespace = "/kobuki";
    // Most people will prefer to do their own velocity smoothing/acceleration limiting.
    // If you wish to utilise kobuki's minimal acceleration limiter, set to true
    parameters.enable_acceleration_limiter = false;
    // If your battery levels are showing significant variance from factory defaults, adjust
thresholds.
    // This will affect the led on the front of the robot as well as when signals are emitted
by the driver.
    parameters.battery_capacity = 16.5;
    parameters.battery_low = 14.0;
    parameters.battery_dangerous = 13.2;

    // initialise - it will throw an exception if parameter validation or initialisation
fails.
    try {
      kobuki.init(parameters);
    } catch ( ecl::StandardException &e ) {
      std::cout << e.what();
    }
  }
private:
  kobuki::Kobuki kobuki;
};

int main() {
```

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

If you compile and run this program, you should hear kobuki's starting up chirp followed immediately by the shutting down chirp.

Library Api

Raw Data

The kobuki driver runs a background thread which continually retrieves packets from the kobuki, saving the raw data, and additionally doing some processing before updating the current state of the robot.

Raw data can be accessed at any time by one of the following getXXX commands:

- **kobuki::Kobuki::getCoreSensorData**
- **kobuki::Kobuki::getDockIRData**
- **kobuki::Kobuki::getCliffData**
- **kobuki::Kobuki::getCurrentData**
- **kobuki::Kobuki::getGpInputData**
- **kobuki::Kobuki::getInertiaData**
- **kobuki::Kobuki::getRawInertiaData**

The gyro provides both filtered yaw angle as well as unfiltered 3-axis inertial data hence the two calls above.

Processed Data and Status

The following are convenience methods for accessing the current state of the robot

- **kobuki::Kobuki::batteryStatus**
- **kobuki::Kobuki::getHeading**
- **kobuki::Kobuki::getAngularVelocity**
- **kobuki::Kobuki::versionInfo** : hardware, firmware and software version strings.
- **kobuki::Kobuki::getWheelJointStates**
- **kobuki::Kobuki::isAlive** : true if a kobuki is detected and streaming data.
- **kobuki::Kobuki::isEnabled** : true if the motor power is enabled
- **kobuki::Kobuki::isShutdown** : true if the worker threads for this driver have been shut down.

Soft Commands

- **kobuki::Kobuki::resetOdometry**

Hard Commands

- **kobuki::Kobuki::setBaseControl**
- **kobuki::Kobuki::setLed**
- **kobuki::Kobuki::setDigitalOutput**
- **kobuki::Kobuki::setExternalPower**
- **kobuki::Kobuki::playSoundSequence**

The Differential Drive Module

The final function of importance is the **kobuki::Kobuki::updateOdometry** method. This updates the current odometry state of the robot, fusing encoder and gyro heading data with the previous known state. For deterministic odometry, it is important this method is called each time a new data packet from the kobuki arrives. Refer to the **simple control loop example** for more information and working

code to illustrate.

---

kobuki_driver
Author(s): Daniel Stonier , Younghun Ju , Jorge Santos Simon
autogenerated on Wed Sep 11 2013 17:03:53