Embedded Software Development for a Mobile Robot

Ondřej Staněk
www.ostan.cz
SRH Hochschule Heidelberg, Master IT, Project work

July 24, 2013

Abstract

Robotic competitions are exciting events that motivate students to gather together and develop a robot collectively. This is a great opportunity for students to learn how to work in a team and also it lays out requirements for systematic and documented work that other teamworkers could build on.

During my study stay on SRH Hochschule Heidelberg, I was the leader of a team of students building robot for the Eurobot 2013 competition. This paper describes the software part of the project as well as the tools we used for development and collaboration. Together we worked on an embedded software system of the Hanuman robot. The task was to develop all the software from scratch, including the low-level drivers for sensors and actuators.

In the Design part, I suggest several programming guidelines that would help the team members to develop easily understandable and maintainable code. It was important to establish a solid framework, choose right development tools and lay out guidelines that other programmers could stick to.

The implementation part then describes several interesting aspects of our embedded system, such as Bluetooth communication and a universal terminal interface for robot diagnostic and teleoperation.

Part I

Analysis

Chapter 1

Introduction

Robots are created to help people in many diverse activities and procedures. In some industries, the automation reached a level where basically all manual workers can be replaced with robots, which are faster, more reliable and effective. Robotisation in the industry has happened quietly and almost without the notice of general public, but recently, robots infiltrate into our daily lives: Automated robotic vacuum cleaners became a common article in electro stores, as well as robotic toys. The state-of-the-art automobiles are equipped with auto-parking feature, which finds a parking spot and parks the vehicle automatically. It is sure that people will face automation more and more frequently in near future.

1.1 Mobile Robots

Mobile robots are automatic machines that can move in their environment. They are usually equipped with an embedded system that evaluates data from robot's sensors and performs specific tasks. Every mobile robot has some means for sensing and affecting its surroundings. These are called *sensors* and *actuators*.

1.2 Sensors

Sensors provide the robot with information about its environment. Thanks to data from sensors, the robot can perceive the world around it. Basically, sensors are for a robot the same what senses are for a human.

There are many types of sensors, from the simplest ones like contact switches

for detecting collisions to more advanced sonars and laser scanners. The robot can be equipped with a camera, microphone, GPS, compass, accelerometer, gyroscope and dozens of others. In short, any measuring device can be a sensor for a robot.

1.3 Actuators

The robot has some means for affecting its surroundings as well. These are called actuators. The essential actuators of every mobile robot are the mechanisms of its locomotion. For instance, the motors that propel robot's wheels, legs, propeller or whatever.

Further, the robot can be equipped with some kind of robotic arm for manipulating objects, or it can carry a vacuum cleaner or a gun. The possibilities are really wide, everything what affects robot's environment in some way is an actuator.

1.4 The Eurobot Contest

Robotic competitions introduce every year new challenges for teams of students and individuals. One of these competitions is Eurobot, an international amateur robotics contest, organized since 1998. It is aimed mainly to graduate students in technical fields. Every year, new rules are published and new game elements are presented. This year (2013), robots are supposed to celebrate their birthday. Robots must unwrap gifts in order to reveal their content, they should serve drinks to guests, put cherries on the cake and last but not least, blow out as many candles on the cake as possible.

Our goal is to develop a robot for the Eurobot 2013 competition. To succeed in a complex software team project, it is necessary to establish some baseline and working style that the team members could stick to. This will be described in the following part.

Part II

Design

Chapter 2

Development Tools

Since the Hanuman robot is a team project, it was very important to establish a solid development environment and unified coding style that will allow to keep the project well organized and maintainable. It was assumed that several programmers will work on the same code. Therefore, I set up collaboration tools (code repository server) as well as documentation Wiki system hosted on GoogleCode [3].

2.1 Source Control Management Tool

Source control management tools help significantly to organize source code that is maintained by many programmers simultaneously. First, it helps to synchronize and merge work of programmers that are editing the same code. Furthermore, it visually shows all the changes done by others, so the developer can easily take track of what has been changed in the code and why. When a programmer makes a change in the code, he is then requested to commit the change for others to see. When doing the so called "commit", the programmer describes new features he implemented or bugfixes he made in the code. That way, the development is always well documented.

But there is many more advantages behind a revision control system. The revision control system keeps track of all changes that have been made to the code, so if some change shows to be fatal and destructive, the revision control system allows to fall back to a working revision. So it also stands for very good backup tool for the source code. That is the reason why revision control systems

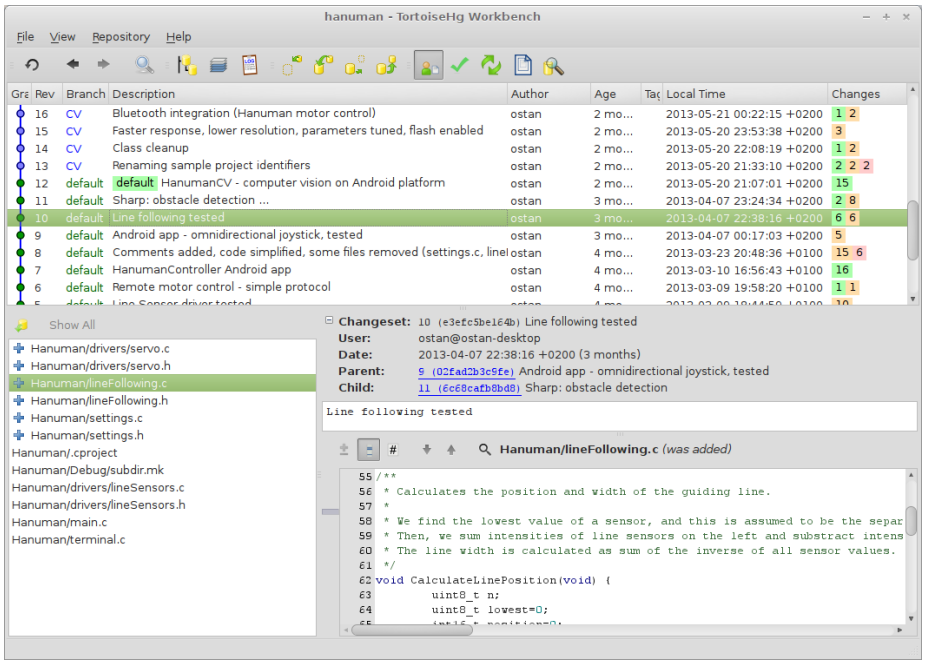

Figure 2.1: TortoiseHg GUI - Hanuman revision graph

are often used even by a single programmers, when no collaboration is expected.

For the project I chose the Mercurial repository system [5]. Mercurial is an free distributed source control management tool, that is easy to learn, yet very powerful. Mercurial has nice graphical interfaces for Windows and Linux systems, that greatly simplify the revision control, as well as code diffs and merges. The suggested Mercurial GUI is TortoiseHg [6]. The convenient graphic interface of TortoiseHg and Meld diff tool is shown on Figure 2.1 and 2.2 respectively.

2.2 Teamwork

Every programmer got a specific task to solve, as for example to develop a driver for a specific peripheral or sensor. To keep the programming work well structured, I suggested particular interfaces of the drivers (function prototypes) to be implemented. Since the specification of interfaces was clear from the very beginning, it allowed other team members to comprehend the function prototypes in their code and thus this eliminated the unnecessary delays in development, since one programmer didn't have to wait until his colleague implements a func-

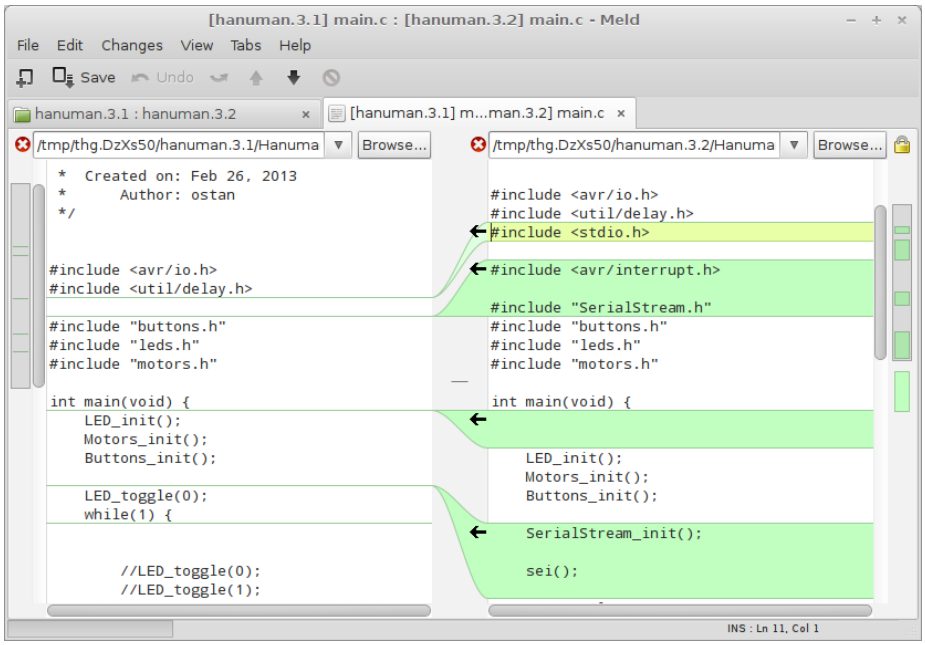

Figure 2.2: Meld - Graphical source code diff

tion he is going to use in his code. He could just use a prototype meanwhile, until his colleague commits the tested driver into the source code repository.

2.3 Toolchain and IDE for Embedded Software Development

The selection of a toolchain was straightforward, for AVR microcontrollers there is a good multi-platform toolchain AVR-GCC [7]. On Windows system, it is supplied together with AVR Studio [8] or WinAVR [9]. The toolchain is used to compile and link a C code to an binary `.hex` file, which is then uploaded to the target (AVR MCU in Hanuman robot). For the uploading of new firmware there is a command-line tool `avrdude` [10]. As a hardware programming tool we used the STK500v2 clone. The uploading procedure is done by invoking a command:

```
$ avrdude -p m128 -c stk500v2 -P /dev/ttyACM0 -U flash:w:Hanuman.hex:a
```

However, the uploading of new firmware can be done directly from IDE, using a button. I suggested Eclipse as an IDE for the development. Eclipse is an

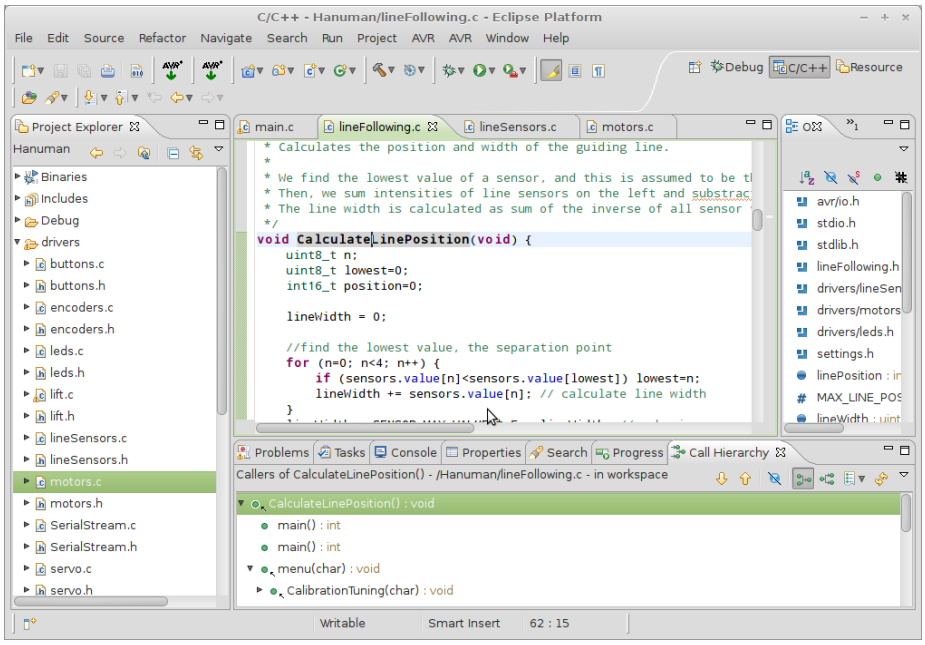

Figure 2.3: Eclipse IDE

open-source multiplatform tool, easy to learn, but with high potential in terms of code refactorization, analysis and debugging. The Eclipse can be configured to support AVR microcontrollers by installing the AVR-Eclipse plugin [11]. With my guidelines we setup the development tools for every programmer in the team.

Chapter 3

Programming Guidelines

In a team, it is important to agree on a common programming style, so that the code is maintainable by all team members. In this chapter I present several design templates and paradigms that will set a framework for every programmer in the team.

3.1 Embedded Control System of a Robot

Program in a robot controls robot's sensors and actuators and it performs the high-level tasks required for the Eurobot competition. The program runs in an 8-bit microcontroller on board of the robot, and therefore it is referred to as an Embedded System. The processing power of the microcontroller is limited, which has to be always considered when developing a software for such platforms. Programming 8-bit microcontrollers is very specific, because there is no operation system that guarantees fundamental features like threading, device drivers or file system. The programmer has to ensure required functionality on his own. Therefore, I point out guidelines for designing peripheral drivers and modules in a microcontroller, as well as some programming practices specific for microcontroller programming.

Since the Eurobot project is a teamwork, it is very important to set some rules and framework so that all team members can stick to that and have a solid base for development. Here, I present one of possible paradigms that results in a readable and maintainable code, unifies the debugging and clearly states the interfaces of the drivers and modules that will be developed by the

team members.

3.2 Microcontroller

Microcontroller (abbreviated MCU) is a single-chip device, that can run a computer program with a minimum of external components. It is a small computer fitted into an integrated circuit. Besides the processor core, the microcontroller usually contains volatile operation memory (typically SRAM[1]), non-volatile program memory (typically FLASH[2]) and peripherals for interfacing other devices.

Microcontrollers are intended for use in embedded systems, such as consumer electronics, remote controls, automobile industry, toys and others.. For these applications, objectives like low power consumption, durability or cheap price are preferred over computational performance. With respect to low processing power, the programmer should design the software carefully.

3.3 Modules and Peripheral Drivers

The microcontroller is equipped with peripherals which allow interfacing sensors and actuators. To list a few of them, such peripherals include: General Purpose Input/Output (GPIO), Analog-to-Digital Converter (ADC), Universal Synchronous/Asynchronous Receiver/Transmitter (USART), I2C[3] serial bus, Timers with Output Compare feature for Pulse-Width Modulation (PWM) and many others..

All these peripherals need to be configured prior usage. For every sensor and actuator there must be a driver that configures appropriate peripherals and provides the interface for the particular sensor/actuator. The driver should carry out:

- Initialization of peripherals for interfacing sensor/actuator
- Configuration of the sensor/actuator (optional)
- Data retrieval from the sensor / Driving the actuator

Drivers provide basic interface to sensors and actuators. Modules make use of these interfaces to provide additional features and perform more complicated

[1] Static Random Access Memory
[2] non-volatile memory that can be erased and reprogrammed in units of memory called blocks
[3] Inter-Integrated Circuit

tasks. In our project, a good example of a driver is a code for motor control. The motor driver configures the timer peripheral for PWM generation:

`void Motors_init(void);`

and provides function for setting the power applied to motors:

`void SetMotorPower(uint8_t motor, int16_t speed);`

Please refer to `/drivers/motor.c` for source code documentation.

An example of a module is the line following code, located in `/lineFollowing.c`. This code gets information about the line position from the lineSensor driver and then sets the appropriate speed for robot's wheels using the motor driver.

It is a good practice that every driver/module is in separate source file and provides unified interface for initialization - *`customDriver`*`_init()`. Optionally, if the driver/module requires executing operational routines on time regular basis, the *`customModule`*`_task()` is designated for this purpose. Of course, the driver or module will introduce other specific functions for driving actuators and reading sensor values. It is crucial to implement these functions as non-blocking whenever possible. The use of blocking functions for sensor data retrieval significantly decreases the performance of the program.

3.3.1 Example of Non-blocking Peripheral Driver

Good example of the non-blocking implementation is a driver that transmits and receives data through the UART communication interface (`/drivers/uart.c`). The UART peripheral is only capable of sending and receiving the data on a single-byte basis. When sending a byte, the driver copies the byte to specific register of UART peripheral and initiates byte transfer. The byte transfer takes quite a long time, and when it is complete, the driver is notified. The programmer often needs to sent a sequence of bytes (text string or a data packet). A naive implementation of the UART driver would be a blocking `sendString(String)` function that sends individual bytes one after another and waits until the last byte was sent. But this is way too much ineffective, as UART transmission is orders of magnitudes slower than CPU! The non-blocking implementation of `sendString(String)` would rather copy the data to a *circular buffer*, from where the data is sent later, in an asynchronous manner, using the interrupt techniques. For receiving bytes, similar interrupt approach must be used, as no other way is reasonable in that case.

3.3.2 Debugging and Error Messages

It is a good practice to output textual debugging and error messages from modules, which will make the development and debugging easier. A unified debugging interface with adjustable verbosity levels will be used. This method was introduced in Dean's Camera LUFA project [2].

The `printf` function provides convenient way for printing strings and numbers in human readable form. First, it is necessary to setup the C `printf` function to work properly on a microcontroller. Obviously, there is no screen, so the output will be passed to a serial interface and displayed in PC. Any stream device can be used as output for the `printf` function. The programmer only need to set up `printf`'s TxByte hook accordingly.

Then, in a header file of a module, there will be a macro providing debugging output with adjustable verbosity levels. See Listing 3.1.

```
#define PROTOCOL_DEBUG(l, s, ...) do { \
    if (PROTOCOL_DEBUG_LEVEL >= l) \
    printf_P(PSTR("(PROTOCOL) " s "\r\n"), ##__VA_ARGS__)
        ; \
    } while (0)
#define PROTOCOL_DEBUG_LEVEL 1
```

Listing 3.1: Debugging macro definition

And the usage of macro will be as follows:

```
PROTOCOL_DEBUG(1,"ProcessPacket: wrong checksum, 
    discarding packet");
PROTOCOL_DEBUG(2,"Received checksum: %d, Expected: %d",
    packet->checksum, sum);
```

Listing 3.2: Debugging macro - example of usage

This method of textual debugging is flexible and easy to manage. It guarantees visually pleasing and unified debugging output for diverse modules, and can be easily turned off in the release version of the software.

It is important to mention that `printf` might be an overkill in some usage scenarios. In addition, because `printf` function is time-consuming and non-reentrant, the debugging macros cannot be used in interrupt routines. We use the debugging macros in places that are not time critical, e.g. for informing about initialization of modules/drivers, or for signaling unrecoverable error states, where the use of time-consuming function does not matter any longer.

Chapter 4

Communication Interface

To see what is happening inside the embedded system (robot), we need to have a way how to retrieve some information from the system. For this, we use an UART communication interface on the AVR MCU. The communication with the robot is crucial especially during development and debugging, when we need to verify the output of sensors and state of the algorithms.

4.1 Wireless Connection to the Robot

It is very useful to have a wireless link to the robot, which can be used for remote control as well as for debugging. Bluetooth is a great technology that can be added to any existing robot very easily and at a low cost, by using the existing UART interface.

4.1.1 Bluetooth

Bluetooth is a wireless standard for interconnecting mobile devices. It is intended for forming ad-hoc Personal Area Networks (PAN). The effective range is approximately 10 meters for a regular (Class 2) Bluetooth device and 100 meters for Class 1 device.

The Bluetooth technology is designed with ease of use in mind. It provides mechanisms for convenient device and service discovery. Establishment and configuration of a Bluetooth connection is easy for the application user.

The data rate can be as high as 3 Mbit/s for the Bluetooth 2.0 + EDR[1] ver-

[1] Enhanced Data-Rate

sion. The Bluetooth technology supports various data transport modes, suitable for transferring both synchronous and asynchronous data. Finally, the Bluetooth technology is supported basically in every cell phone and laptop, which is a great advantage.

4.1.2 Bluetooth RFCOMM modules

There are two ways how to implement Bluetooth technology in a robot. The Bluetooth stack can be located either in the Bluetooth module itself, or it is implemented in the microcontroller of the robot. The first option is commonly used in robotics. Implementing a Bluetooth stack in a microcontroller is a *very* challenging task, so many authors rather choose single-purpose Bluetooth modules that have protocol stack integrated on chip. Such modules are known as Bluetooth RFCOMM modules and they are meant as a cable replacement for the Serial Port (RS232). When the RFCOMM module is linked with a computer, the operation system creates a virtual serial (COM) port. The data stream sent to the virtual port is transparently carried over Bluetooth channel to the serial interface of robot's microcontroller. This allows to build up a wireless connection to the robot with minimal effort. However, the capabilities of integrated Bluetooth stack are very limited; basically only a single point-to-point RFCOMM connection initiated by the host (computer or phone) is possible. The configuration of such module is limited as well.

For our robot, I decided to get a cheap[2] RFCOMM module from DealExtreme.com [12]. This module can be connected directly to the UART interface of the microcontroller, and it is very nice replacement for cable. With a minimal effort, a universal Bluetooth interface was added to the robot. This Bluetooth technology allows the robot to be controlled from a PC as well as from a smart phone.

4.2 The Terminal Interface of the Robot

Inspired by effectiveness of Linux-like terminals, I decided to develop a terminal interface for the Hanuman robot that would allow to access and configure all the drivers and modules of the robot. The terminal interface is solely textual (ASCII) oriented, so anyone can access the robot interface without need of custom control application. Any serial terminal will do the job. The terminal

[2] RFCOMM module costs about 8 USD

Figure 4.1: RFCOMM Bluetooth module from DealExtreme

interface is supported on all operation systems, including Android OS for smart phones. Therefore, the robot can be controlled from any device without need to install any particular software on the device. Figure 4.2 shows the terminal to Hanuman opened from a Linux system.

On Linux based systems, the the terminal to Hanuman robot can be opened issuing the two commands in a Unix shell.

First, we establish a Bluetooth connection to the RFCOMM module on the robot. For this, the MAC address of the module has to be specified.

```
$ sudo rfcomm connect rfcomm5 00:13:03:26:14:01
```

Afterwards (in another shell) we open the terminal program. Please note that the communication baudrate is set to 9600 bauds:

```
$ screen /dev/rfcomm5 9600
```

After pressing any key the welcome screen of Hanuman terminal will appear, as shown in Figure 4.2. The user then select a number of the specific category he or she wants to enter.

On Android Phone, the pairing procedure has to be invoked before the first connection to the Hanuman robot. The PIN is `1234`. Then, the BlueTerm application can be used to establish the connection to the Hanuman robot. This is shown in Figure 4.3.

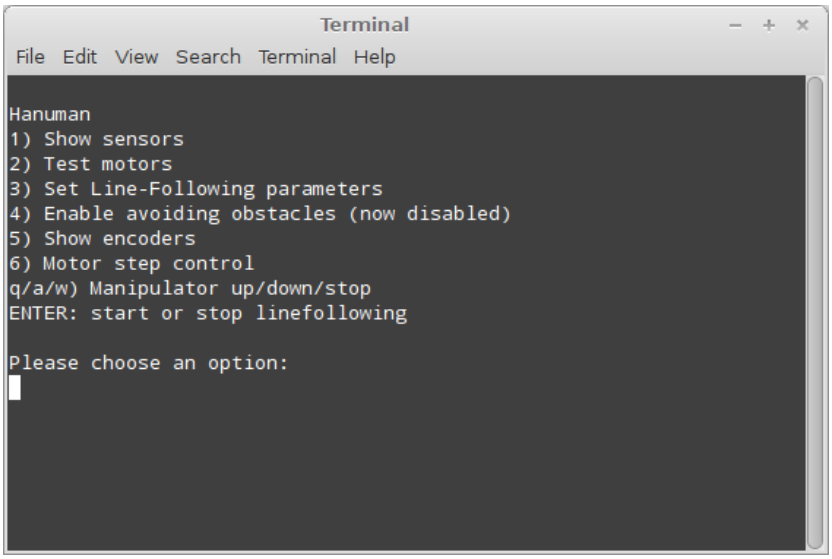

Figure 4.2: Terminal to the Hanuman robot in Linux system

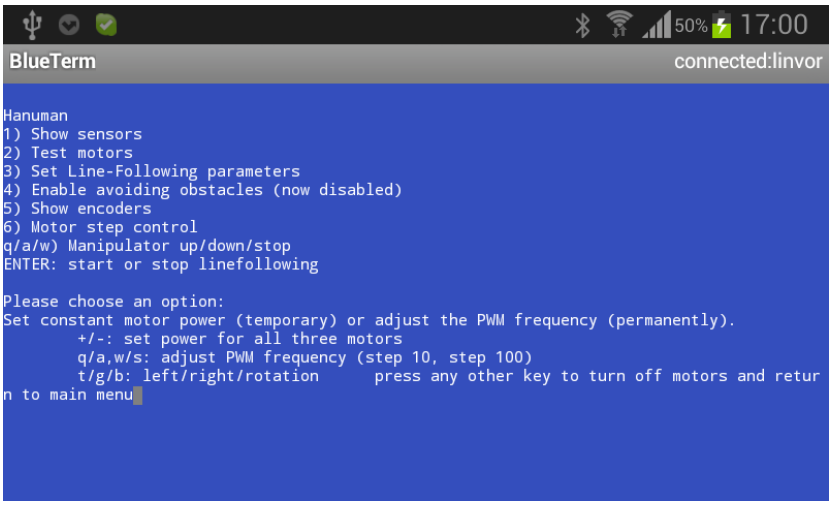

Figure 4.3: Terminal to the Hanuman robot in Android OS

Part III

Implementation

Chapter 5

Hanuman Robot

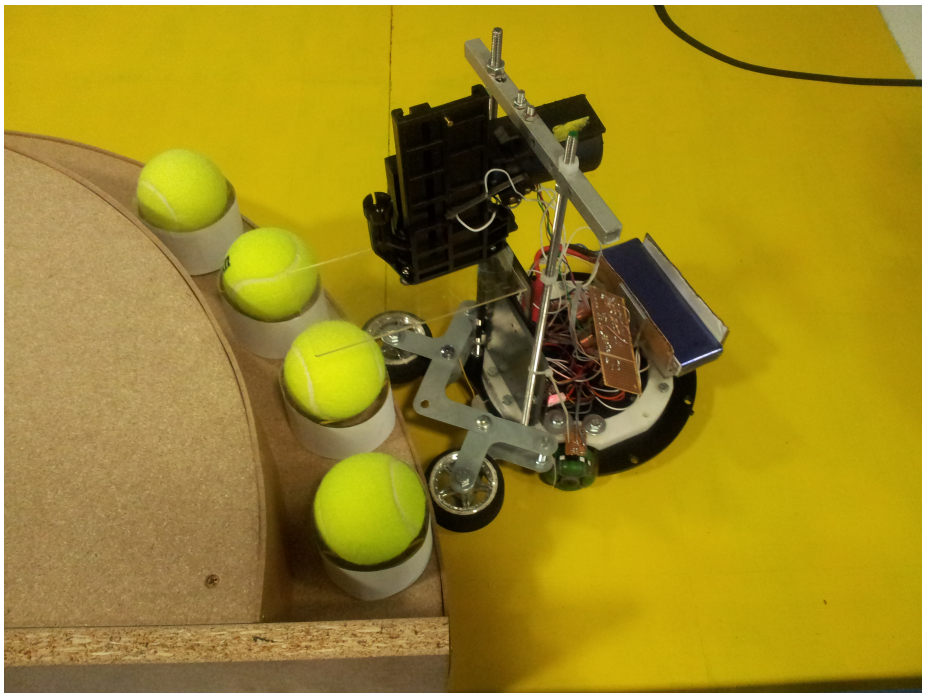

Figure 5.1: Hanuman robot - performing tennis ball manipulation

5.1 Mechanics

The Hanuman robot has an omni-directional wheelframe. The three wheels allow movement in any arbitrary direction, or rotation in place. That makes the platform very flexible for solving different tasks. In front of the robot, there is a manipulator that pushes the tennis balls inside their supporting cylinders. Robot is equipped with a holder for a smart phone. The smart phone camera looks straight forward underneath the ball manipulator.

5.2 Electronics

The robot is controlled with an embedded system. On board, there is an ATmega128 8-bit microcontroller and supporting electronic circuity for controlling the motors. The robot is equipped with a standard Bluetooth serial RFCOMM adapter. This adapter plays the key role in communication with the smart phone, which can run more complex control algorithms.

5.3 Firmware

The ATmega128 microcontroller is programmed in C language. The firmware implements drivers for motors, encoders, ball manipulator and other sensors. It also provides a useful service menu that can be accessed from any computer or smart phone via Bluetooth. Finally, it supports binary protocol for communication with the Android phone.

Chapter 6

Embedded System Implementation

There is many drivers and modules that have been implemented in the Hanuman firmware:

- motor control (`/drivers/motors.c`)
- reading wheel encoder (`/drivers/encoders.c`)
- feedback wheel control driver (`/drivers/wheelStepControl.c`)
- reading line sensor with ADC (`/drivers/lineSensors.c`)
- reading button state (`/drivers/buttons.c`)
- signalling LEDs (`/drivers/leds.c`)
- controlling lift manipulator (`/drivers/lift.c`)
- serial communication (`/drivers/SerialStream.c`, `drivers/uart.c`)
- line following (`/lineFollowing.c`)
- terminal interface (`/terminal.c`)

The robot can be controlled from terminal, and the state of sensors can be displayed and analyzed through the text interface on UART. There is an Android application that can control the speed of motors by sending control packets to the UART interface of robot. The board provides an interface for controlling

motor speed, wheel stepping and its lift manipulator via UART/RFCOMM, binary protocol. Please see `drivers/uart.c` how this is implemented.

6.1 Motor Driver

6.1.1 The Motor Driver Interface

The interface of the driver is specified in Listing 6.1. This interface was developed before any actual coding work started, so that other programmers could plan on using the functions that will be implemented. Interface specification is essential for the team collaboration.

```
/**
 * Copyright (C) Ondrej Stanek
 * ostan89@gmail.com
 * http://www.ostan.cz
 *
 * Driver for motors, uses TIMER1 for PWM generation.
 *
*/

/**
 * configures TIMER1 for PWM generation
 */
void Motors_init(void);

/**
 * Sets the PWM duty and direction of motor
 * @param motor
 *  Sets duty for motor id 0, 1 or 2
 * @param speed
 *  The speed ranges from -255 to 255
 *  speed = 0 means that the motor stops
 */
void SetMotorPower(uint8_t motor, int16_t speed);

/**
 * Stop all motors immediately
```

```
 */
void StopMotors(void);

/**
 * Adjusts the PWM frequency, can be changed in real time
     during operation
 */
void setMotorPWMfrequency(uint16_t frequency);
```

Listing 6.1: Interface of the motor driver

6.1.2 Motor Driver Implementation

The motor driver uses a 16-bit Timer/Counter for PWM generation. The timer/counter peripheral is configured in the initialization phase, when the function `Motors_init()` is called. This function configures the TIMER1 peripheral for operation in Phase and Frequency Correct PWM mode, with three Output Compare Channels. It also configures output pins for setting the motor direction.

Function `SetMotorPower(uint8_t motor, int16_t speed)` then sets the direction of rotation (depending on the sign of the `speed` variable) and the PWM duty for appropriate Output Compare channel.

It is possible to adjust the PWM frequency. This is done by changing the `ICR1` register of the `TIMER1` peripheral. The PWM frequency should be set according to the hardware capabilities and motor characteristics.

6.2 Line Following

Hanuman robot is capable of line following. It follows a black line marked on bright surface. According to the Eurobot 2013 rules, the line guides the robot to a gifts that need to be unwrapped.

The line-following algorithm is a simple control loop feedback mechanism [13]. Optical sensors measure the light reflectivity of the surface and acquired data is processed by the line detection algorithm. The algorithm is designed in such a manner that line width does not matter. The line detection algorithm outputs signed integer value that states the actual deflection of a guideline (error term $e(t)$ in time t). Values close to zero mean that the line is located accurately

in the middle of the sensor module, positive values state how much does the line deflects to the right and negative values state the deflection to the left. This output is then used for controlling the line tracking.

6.2.1 PID controller

This section describes the proportional-integral-derivate (PID) controller [14], which is a universal closed loop feedback algorithm. We will describe how the PID controller is utilized to control line tracking.

The PID controller adjusts the wheels' speed according to the actual line deflection and previous states. Defining $u(t)$ as the controller output (difference between speed of left and right wheel), the PID algorithm is as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t)$$

where K_p (Proportional gain), K_i (Integral gain) and K_d (Derivative gain) are parameters of the PID regulator. See PID controller overview in Figure 6.1.

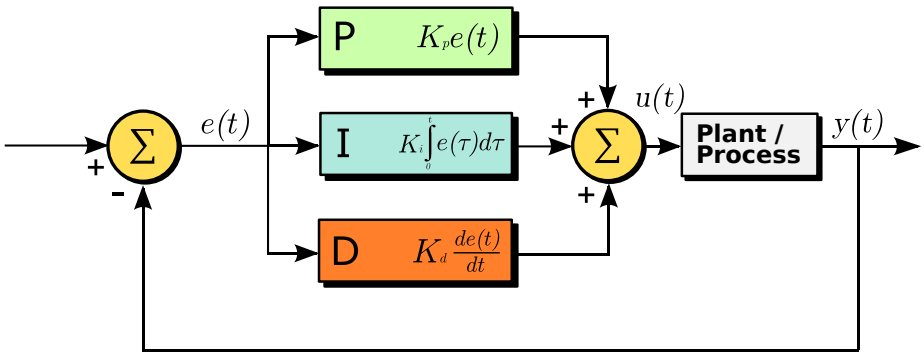

Figure 6.1: PID controller overview

The Proportional term K_p affect wheel speed in a way that the robot turns towards the guiding line. However, at higher speeds, the Proportional regulation is not sufficient, because the system starts to overshoot. Therefore, we set the Derivative term K_d which affects $u(t)$ when the rate of change of error $e(t)$ is considerable. That suppresses overshooting. The Integral part of regulator is not used for line following, so the K_i is set to 0.

In the microcontroller, a discrete version of PID regulator is implemented [15]. We approximate the derivative and integral terms:

$$\frac{d}{dt}e(t) \approx \frac{e(t) - e(t-h)}{h}$$

and

$$\int_0^t e(\tau)d\tau \approx h\sum_{i=0}^{t} e(i)$$

Where $h = 1$ is the time period between discrete samples of error term $e(t)$. The discrete version of PID controller is as follows:

$$u(t) = K_p e(t) + K_i \sum_{i=0}^{t} e(i) + K_d \left(e(t) - e(t-1)\right)$$

To conclude, the PID controller drives the robot so that the line is always centered to the middle of the sensor module, so that the robot performs smooth line following.

6.3 Android Control Application

The HanumanController is a simple Java application for Android phones. It allows to control movement of the robot using a virtual joystick on the touch-screen, to control the rotation of the robot or just to set the individual wheel speeds. Finally, user can turn the autonomous line following mode from the application.

A simple binary protocol is used for the communication between the Android phone and the Hanuman robot.

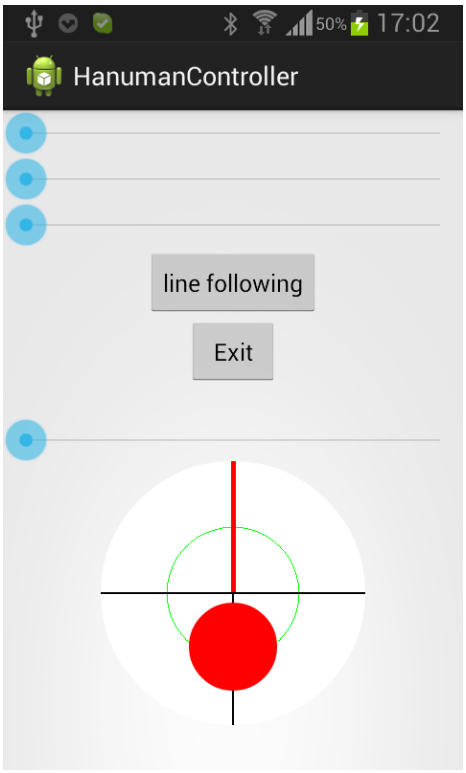

Figure 6.2: HanumanController - Android application

Conclusion

Embedded system for a mobile robot was developed and tested. It supports functionality required by the Eurobot 2013 competition, such as line following and tennis ball manipulation using a specific hardware manipulator.

The whole structure of the Hanuman Embedded System is well designed and logically divided into drivers and modules, that can be developed and tested individually. This is essential when more people work on the same software project. Moreover, a collaboration system was established with an online source code repository (Mercurial, hosted on GoogleCode).

The source code is well commented, mostly using Doxygen annotations, so the documentation can be automatically generated out of the project source files.

A development diary is available online [16].

Bibliography

[1] Ondřej Staněk. Centralized Multi-Robot System, bachelor thesis, The Department of Software Engineering , Charles University in Prague, 2012

[2] Camera Dean. *LUFA (2012)* [online]. http://www.fourwalledcubicle.com/LUFA.php

[3] https://code.google.com/p/hanuman/

[4] https://en.wikipedia.org/wiki/Revision_control

[5] http://mercurial.selenic.com/

[6] http://mercurial.selenic.com/wiki/TortoiseHg

[7] http://gcc.gnu.org/wiki/avr-gcc

[8] http://www.atmel.com/microsite/avr_studio_5/

[9] winavr.sourceforge.net

[10] http://www.nongnu.org/avrdude/

[11] avr-eclipse.sourceforge.net

[12] http://dx.com/p/jy-mcu-arduino-bluetooth-wireless-serial-port-module-104299

[13] Nise S. N. *Control Systems Engineering.* Willey. Fourth Edition. 2004 ISBN 978-0-471-44577-7

[14] *PID controller* [online]. Wikipedia. http://en.wikipedia.org/wiki/PID_controller

[15] Tham M. *Discretised PID Controllers* [online]. http://lorien.ncl.ac.uk/ming/digicont/digimath/dpid1.htm

[16] https://code.google.com/p/hanuman/wiki/DevelopmentDiary