University of New South Wales
School of Computer Science and Engineering
Robotics

**Practicum Exchange Program Thesis**

# Developing a bipedal walk using a Cycloid II

Manuel Lange

| | |
|---|---|
| **Supervisor:** | Prof. Maurice Pagnucco<br>Associate Professor and Head of School of Computer Science and Engineering<br><br>Dr. Bernhard Hengst<br>Senior Research Associate ARC Centre of Excellence for Autonomous Systems, School of Computer Science and Engineering |
| **Started on:** | 01.02.2011 |
| **Ended on:** | 31.07.2011 |

## Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Tübingen am 10. Dezember 2011

______________________

Manuel Lange

**Abstract.** A new walk for humanoid robots which is stable and flexible was to be developed at the robotics area of the School of Computer Science and Engineering of the University of New South Wales, Sydney. The motivation is to use it on the UNSW's RoboCup Soccer team rUNSWift when it works omnidirectional, but first of all the general usability is researched here, wherefore the walking directions are reduced to be bidirectional (forwards/backwards). As the Nao robots of the SPL are not very durable and are likely to break, most of the development and testing was done on a Cycloid II robot.

As part of this research project a Cycloid II robot, which had not been used until then, was set up as a research platform. This included writing the drivers to communicate with the motors and sensors as well as attaching new foot sensors and configuring them to be used. As this first part of the project was finished, the research on the walk where the robot was modeled as an inverted pendulum could be started.

The inverted pendulum model assumes that the robot always has one leg on the ground (the support-leg) and one leg in the air (the swing-leg) where the support-leg is the pendulum's base. As the velocity of the pendulum model is strongly determined by the position of the pendulums base, multiple approaches have been conducted to find out how to control it best. Directly controlling the swing-leg position was one of them but it made the robot hectic and less stable than letting it follow the pendulums natural swing. Better results were achieved when only the ankle tilt of the support-leg was controlled directly to de-/accelerate the swing of the pendulum.

A simulator with an inverted pendulum model of the robot was used to create reinforcement learned (pendulum) controllers for the tests conducted, whereby most of the calculation is done in simulation. During runtime the physical robot only has to estimate its current state as good as possible to look up the optimal action from the pre-learned controller in order to achieve the current goal. The goal can either be to walk on the spot or to walk forwards/backwards while keeping the balance even when disturbed by outer influences.

When the best results were achieved, the reinforcement learned pendulum controller supplied the robot with the control actions determining how to set the ankle tilt. Thereby the robot became able to handle disturbances like stepping on objects or running against them without falling over as well as quickly switching movement directions. These results prove the usefulness of the system investigated in this research project and furthermore they promise successful results when extending it to omnidirectional movement by continued research.

# Acknowledgement

I would like to express my gratitude to the people who helped me during this project. My two partners of our joint paper, containing my research documented herein:
Bernhard Hengst who made this project possible and always had some inspiring ideas whenever I got stuck.
And Brock White with whom I used to exchange experiences.
Additionally I'd like to thank Jason Kulk for his advice in the beginning, while figuring out how to communicate with the Dynamixel actuators.

Furthermore I would like to give my special thanks to my Co-Supervisor Prof. Andreas Zell from my home university - the Eberhard Karls University of Tübingen - for his support while going on this exchange program as well as while integrating it into my degree.

# Contents

**1. Introduction** **1**
1.1. Motivation . . . 1
1.2. Concept . . . 1
1.3. About the Cycloid II . . . 2

**2. Robot and sensors** **4**
2.1. Cycloid II . . . 4
2.1.1. Specifications . . . 4
2.2. Motors . . . 5
2.3. Foot sensors . . . 8
2.4. Inertial measurement unit (IMU) . . . 9

**3. Theoretical principles** **11**
3.1. Pendulum model . . . 11
3.1.1. The reinforcement learner . . . 12
3.1.2. Simulation results . . . 14

**4. Implementation and evaluation** **18**
4.1. Overview . . . 18
4.2. Motion-cycle . . . 18
4.3. Implementation of the controller . . . 20
4.4. Basic rock and walk . . . 21
4.5. Pendulum controller . . . 22
4.6. Threaded read/write using mutex locks . . . 24
4.7. Evaluation by tracking the X . . . 25

**5. Experimental results** **27**
5.1. Overview . . . 27
5.2. Quick policy change . . . 28
5.3. Stepping test . . . 28
5.3.1. Stepping experiment . . . 28
5.3.2. Stepping comparison with and without the pendulum controller . . . 29
5.4. Collision test . . . 30
5.5. Chapter summary . . . 32

**6. Summary and future work** **33**
6.1. Summary and future work . . . 33
6.2. Related work . . . 34

**7. Appendix** **35**
7.1. Continuing research on the Cycloid II . . . 35

7.1.1. Starting the robot . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
7.1.2. Linux system . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
7.1.3. Using the walking code . . . . . . . . . . . . . . . . . . . . . . . . . 36
7.1.4. Known issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
7.1.5. Software used . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

**A. Glossary** **40**

**References** **41**

# 1. Introduction

Robots have been producing goods in factories for years already, but they haven't arrived in everyday life yet. Humanoid robots are built man-like to increase their acceptance in society but since they aren't very useful yet and too expensive, it will still take some more time until they find their way into everyone's household. But as actuators are getting smaller and more powerful as well as microprocessors are constantly improving and are able to handle more complex tasks like vision-processing, they are a big field of research.

Universities are having different robot competitions to enforce research. One example is the Robocup [Roba] where robots are playing soccer against each other or try to support rescuing in disaster scenarios. The industry is developing service robots like the Scitos A5 [sci] to support customers in big stores or shopping centers to find what they want. And also the military is supporting research in robots for example the BigDog[Big], which shall be used to carry supplies through a rough terrain war zone without risking human lives.

Despite the fact that humanoid robots can be found in research facilities since years already, one main problem for humanoid robots still is to walk flexible and to be stable against outer influences, which is addressed here.

## 1.1. Motivation

The robotics area of the School of Computer Science and Engineering of the University of New South Wales wants to develop a new way of walking for their RoboCup Soccer team rUNSWift which is not only fast and flexible but also keeps the robot stable and protects it from falling. A common problem is that the robots step on each others feet or walk into each other. This results in a loss of balance for these robots leading to a fall that can sometimes also take down other robots around them.

Falling over does not only waste important time when playing soccer but it also bears the risk to damage the robot. That is especially a big problem for the Nao robots as they are not very robust and likely to break their joints and other parts fairly regularly when falling over.

Besides that the walk algorithm shall be able to handle being pushed or stepping on small objects, it shall also be able to reverse the robots direction quickly from forwards to backwards and vice versa to be able to react flexibly to every new situation.

## 1.2. Concept

The robot is modeled as an inverted pendulum model and simplified to have all its mass in its center-of-mass, which then equals the point mass of the pendulum. This point mass is connected with two legs where it is assumed that one leg is always in the air (swing-leg) while the other leg is on the ground (support-leg) and both are modeled to be massless. These legs alternate their status (swing-/support-leg) whenever the actual swing-leg moves up and down once whereupon the robot naturally swings sidewards from one to the other leg. With the swing the pressure follows analogously to the other leg. When the pressure changes from one leg to the other, the status of the legs is also changed, which happens with about the natural rocking (sidewards swinging) frequency of the robot. The support-leg is the rod of the pendulum and moves horizontally following the calculated velocity of the pendulum. Whereas the swing-leg is moved opposite in relation to the horizontal distance between the center-of-mass of the robot and the center-of-pressure on the foot of the swing-leg. When the swing- and support-leg change, the position of the rod (base) switches instantly from behind to in front of the center-of-mass or vice versa. Thereby the pendulum is being kept steady.

To control the system's velocity it is made use of the fact that the foot is not just a point but a flat foot with a size. Changing the ankle pitch results in a change of the horizontal position of the center-of-pressure on the foot. Whereby the pivot of the rod of the pendulum can be changed along the size of the foot. This provides the possibility to decelerate or accelerate the pendulum. While the legs only change twice during a rock-cycle, the ankle pitch can be changed many times within each cycle. The robot runs at 100 Hz as the sensors provide new data at that rate and because it is a good rate to make the motors move smooth. One rock-cycle takes about 0.5 seconds which means that the legs only change every 250 ms whereas the ankle pitch can be adjusted every 10ms with each calculation cycle (at 100 Hz). This is a bonus in flexibility to quickly changing situations and to stability when dealing with sudden disturbances.

The idea is not to use an analytic approach where the movement would follow exact formulas but to always do the next smartest move in every cycle of the walk algorithm instead. At first the use of an A* search on the state information of the system was considered to calculate the optimal changes to achieve a targeted goal velocity. But as there are only 10ms time at each cycle that should be used as efficiently as possible, the plan changed from using an A* search to the use of a pre-calculated controller. The inverted pendulum model of the robot is transferred into a simulator that uses machine learning to find the optimal actions for all the states of the modeled pendulum and write them into a controller file. This file is transferred to the robot, which uses the controller to look up the pre-calculated actions related to its actual state. Depending on the implementation the action will tell how to control the ankle pitch and/or the swing-leg stride for each of the states.

Once the system works properly, it should be easy to transfer it to other bipedal robots by adjusting some values in the simulator, like the height and foot size of the robot and generating a suitable controller file. Some testing would have to be done on the robot itself to find its natural rocking frequency and good values for parameters like the height of the leg lift. Limitations could be given by the motors if they are not fast enough or too weak.

## 1.3. About the Cycloid II

The implementation and experiments were carried out on a Cycloid II robot which is made of metal and thereby much more durable than other robots that are made of plastic and also use plastic gears like the Nao robot [Robb]. The joints are also more robust as they don't use any gear wheels outside of the motors.

The Cycloid II is a humanoid robot made by Robotis [rob11], a robot company from South Korea. Tribotix [Tri11] is the robot company located in Newcastle, Australia which sells the Robotis robots in the Australian and New Zealand region to educational institutions as well as to private persons. The UNSW's version of the Cycloid II was upgraded by Tribotix by an additional motor for the tilt movement of the head, a better USB Camera for vision, more space for better batteries and multiple hardware boards equipped with different kinds of sensors and a more powerful processor (see chapter 2).

Coming from Tribotix the robot had one of the latest Linux Debian versions installed. With its 32-bit X86-compatible CPU architecture also Windows could be run on it and is even provided by CompuLab[Com] as Windows XP Embedded - run-time image.

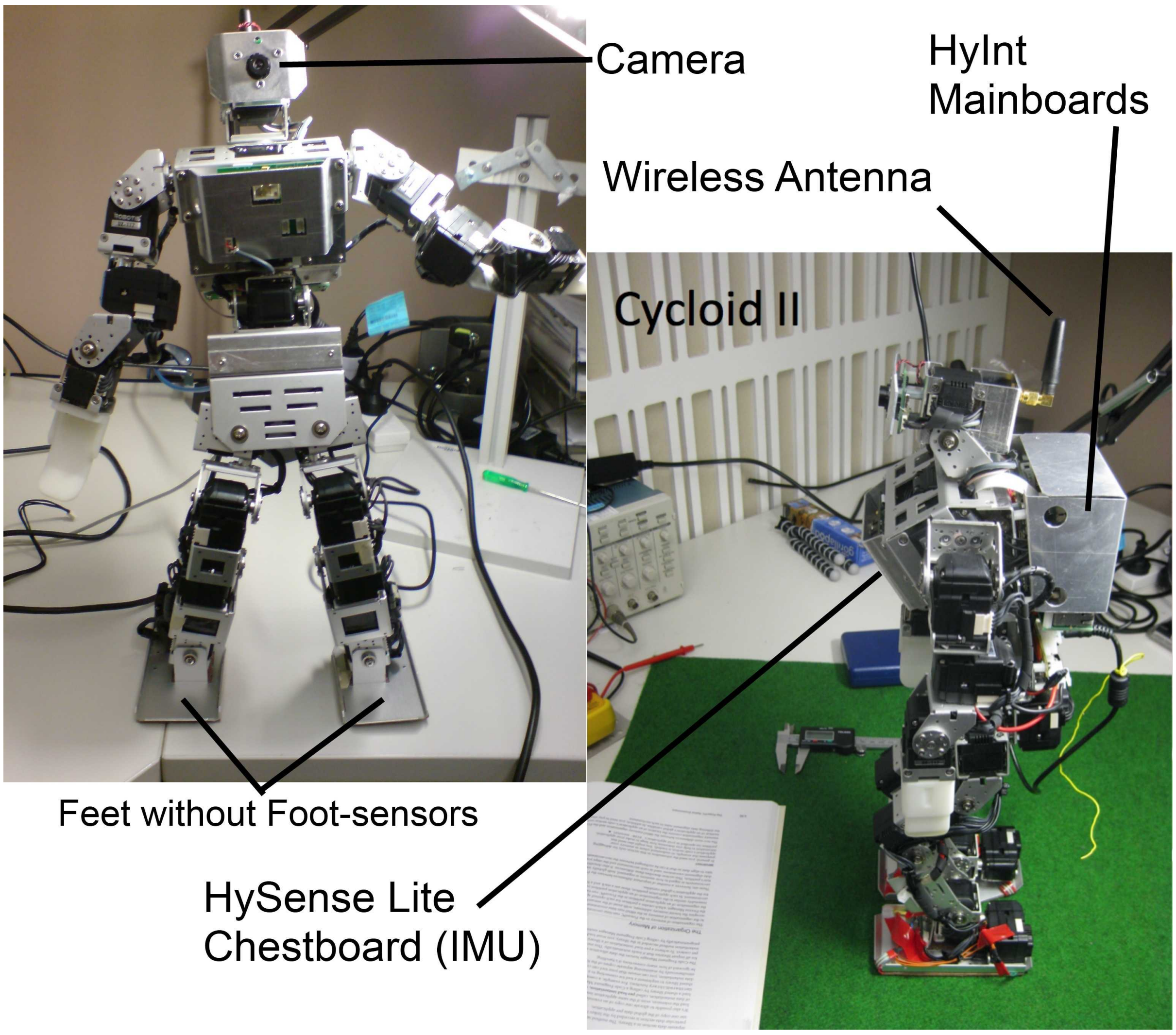


Figure 1.1.: The Cycloid II Robot. Left: a front view picture from the beginning, before the foot sensors were attached. Right: a view from the right.

# 2. Robot and sensors

## 2.1. Cycloid II

As there is no manual for this upgraded version of the Cycloid II, a short overview is given here. The skeleton of the Cycloid II consists of 24 Dynamixel actuators, which are connected through different shaped frame parts. The main powering and processing components are all concentrated in the upper body, which has been reshaped to fit in the upgraded modules. A USB Camera, which is not used in this project, is located in the head of the robot and foot-sensors have additionally been attached to each foot of the robot in the beginning of this project.

### 2.1.1. Specifications

- **Processor:** AMD Geode LXMCU 500MHz x86
- **Memory:** 256Mb DDR SRAM, 512Mb Flash Disk and 16GByte USB 2.0 Flash Memory
- **Ports:** 2 · USB 2.0, RS-232 & RS-485 Serial-Ports, VGA, 100Mbit Ethernet, Mini PCI-bus, 802.11g WLAN
- **Measurements (standing):** Height: 475mm, Width: 200mm, Length: 150mm
- **Weight:** about 3kg (including Batteries)
- **Actuators:** 24 Dynamixel - 23 · DX-117 and 1 · DX-113 from Robotis
- **Footsensors:** One Kit per foot, one sensor-board and four sensor-pads per kit.
- **Batteries:** 2 x 7,4V 2480mAh 10.5C Lithium Polymer Batteries from Tiger Elite (TE554590-2480-2S1P) which last between one and two hours, depending on usage. Charging time with the supplied charger in auto mode takes about 1.5 hours for one battery pack.
- USB-Connected simple VGA-Camera

**Additionally the following modules are attached:**

**HyInt** The HyInt module provided by Tribotix consists of four boards.
The **CM-iGLX board** is supplied by Compulab [Com09] and equipped with an AMD Geode LXMCU 500MHz x86, 256Mb DDR SRAM, a 512Mb Flash Disk and various interfaces like RS 232 & RS-485 Serial-Ports, 100Mbit Ethernet, a VGA port, a WLAN 802.11g Interface, a Mini PCI-Bus and USB-Ports.

The **Motherboard (PB0801)** is stacked on top of the CM-iGLX board providing connectors to all the interfaces supplied by the CM-iGLX board. Furthermore it provides push-buttons for power down and rest options and jumpers to select either TTL or RS-485 for communication. Additionally it connects to the **External I/O Board (PB0804)**, which is only used to lead the VGA, Serial, Ethernet and one USB-Port out of the chest case cover via a flat ribbon cable. But the robot can be run without it.

A **Power Supply Board (PB0803)** is stacked on top of the motherboard to supply all the other components with power by transforming the Voltage coming from either the battery pack or the DC jack to 3.3V/5V and monitoring the power consumption using an ATMega8. It also provides the ports forwarded from the motherboard to connect the motors (RS485) and the foot-sensors (TTL).

**HySense Lite** The HySense Lite module is an optional sensor board, which got attached on the chest of the Cycloid II and connects to the motherboard of the HyInt module via a RS-485 channel provided by Tribotix. It uses an Atmel ATMega128 microcontroller to access its sensors' data and send it to the HyInt module when requested. It is equipped with a 5-axis IMU + a 1-axis Gyro, connectors for IR Distance Sensors, an analog to digital converter and three LED's.

## 2.2. Motors

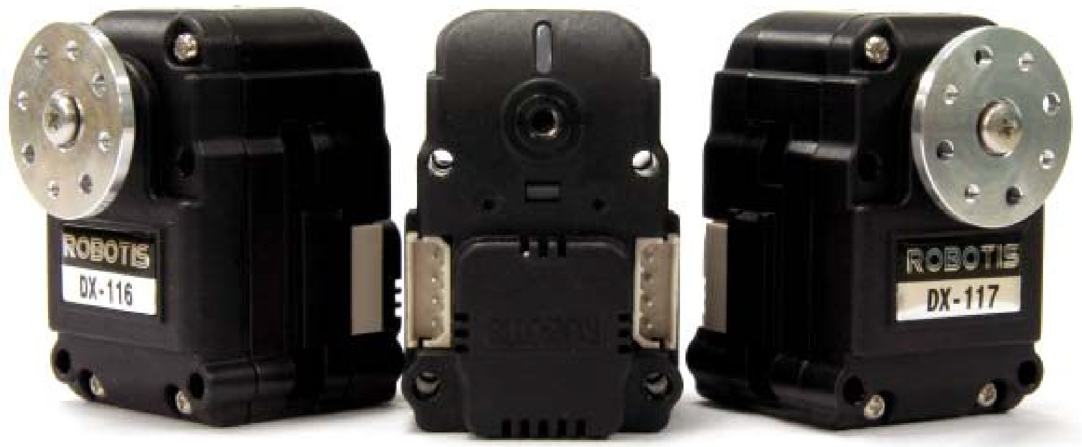


Figure 2.1.: The Dynamixel Series: DX-116, DX-113 and DX-117 all with the same outer appearance, figure from the Dynamixel manual [Rob05]

The Cycloid II is driven by 24 Dynamixel actuators made by Robotix. According to the manual [Rob05] the 23 DX-117 can operate with a Voltage in the range of 12 V to 16 V and have an operating angle of 300° in which they turn with 0.129-0.172sec/60° (no-load Speed depending on Voltage) if not blocked by anything. They have a minimal holding torque of 28.9 kgf/cm with a max. current of 1200 mA, which makes them strong enough to constantly move the cycloid without overheating. Every Dynamixel has its own MCU which makes controlling them simpler as they can receive commands and send answers to status requests through a half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity). The position can be set in a range between 0 and 1023 thereby the resolution is 300° / 1024 $\approx$ 0.29°. Each Dynamixel can be assigned with an ID in the range of 0 to 253 making it possible to chain up to 254 Dynamixels together on the same port with unique IDs, although problems will come up when communication messages overlap. Therefore it is possible to set a return delay time of up to 255 microseconds which is waited for, before the answer to a request is sent.

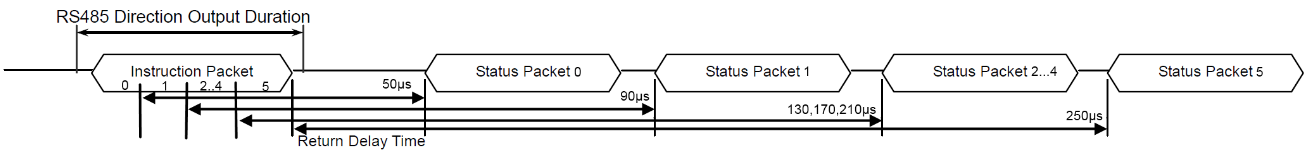


Figure 2.2.: The figure shows an example of the return delay time between the packets when requesting values like the motor positions from the motors. First a set of instruction packets, including a request for each motor, is sent followed by the answers of the motors coming one after the other according to the predefined Return Delay Time. The original figure from the manual [Rob05] has been extended for clarification.

Using the maximum communication speed of 1Mbps, assigning the first actuator a return delay of 50 microseconds and increasing the delay by another 40 microseconds for every additional motor, offers the option to request answers from up to six motors at once and to receive their answers without any overlap as shown in figure 2.2. This allows a timeframe of 50 $\mu s$ for the set of instruction packets to be sent to the motors and a frame of 40 $\mu s$ for each motors answer. This is enough time to prevent the packtes from overlapping each other. Given a time of 10 milliseconds per cycle up to 12 actuators can be controlled over one port by having one thread which requests, waits and reads from the first six motors and afterwards requests and reads from the second bunch of motors. Even when the answers are coming in after $\mu s$ the system takes some ms till writing to and reading from the USB2Serial device is finished. Because of this and due to the maximum return delay time being limited to 255 $\mu s$, it is only possible to communicate with two bunches of six motors each summing up to 12 motors in total over one serial port at the same time when all communication has to be done within 10ms cycle time.

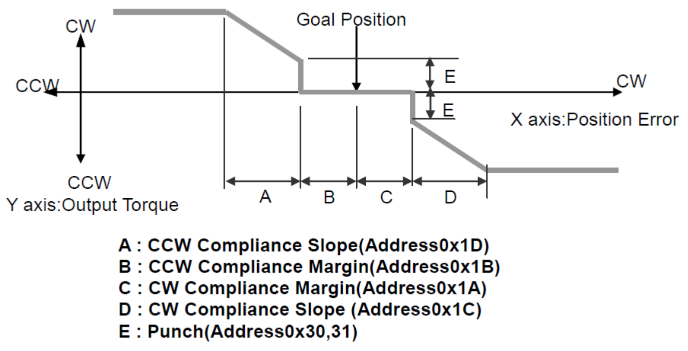
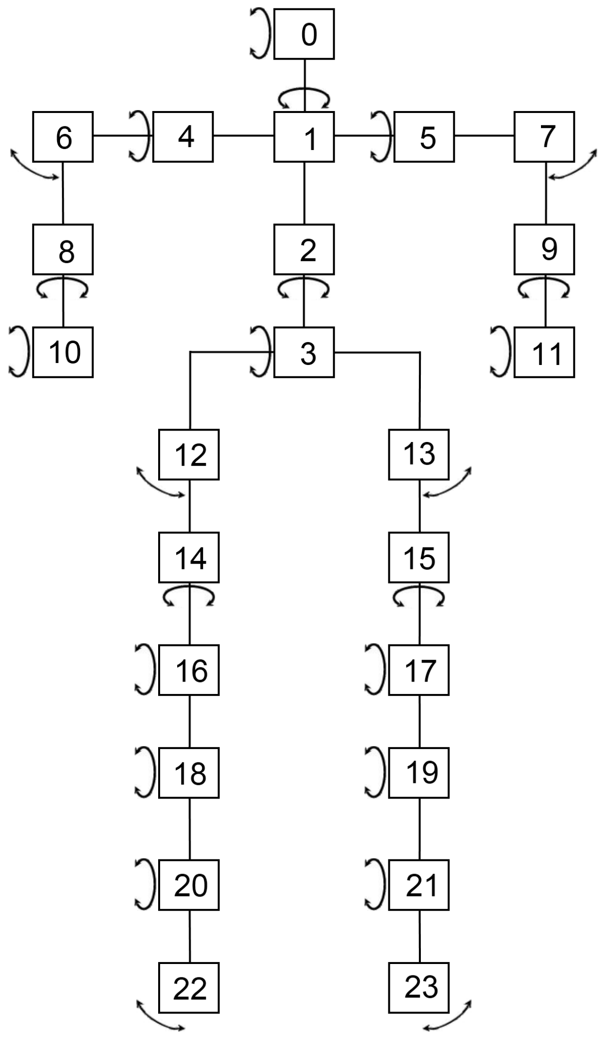

CW
Goal Position
E
CCW
CW
E
X axis:Position Error
CCW
Y axis:Output Torque
A
B
C
D
A : CCW Compliance Slope(Address0x1D)
B : CCW Compliance Margin(Address0x1B)
C : CW Compliance Margin(Address0x1A)
D : CW Compliance Slope (Address0x1C)
E : Punch(Address0x30,31)

Figure 2.3.: The area of the slope and margin in each direction influence the behaviour of the robot. Smaller margin and slope values make the robot less wobbly but if they are set too small, it gets shaky. [Rob05]

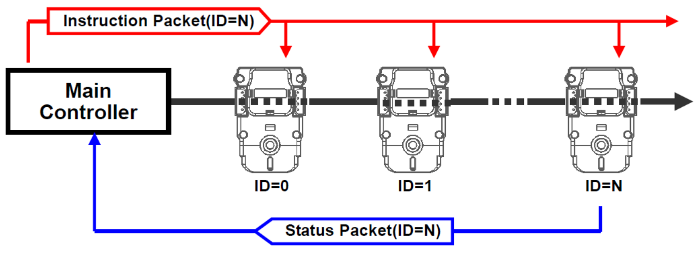
Most of the initial values for the motors' settings are already fine for the use in the Cycloid II.

But to get some slack out of the machine, some settings have proven to be important to change. To be in compliance with the goal position set to a motor, it has a margin around that position in which it doesn't put any torque on the motor. On both sides around that margin is an area of slope in which the output torque increases, the further the current position of the motor is away from the goal position. It increases up to the maximum torque which is at the end of the slope area, shown in figure 2.3. Coming from Tribotix the motors' values for the margins were fine but the slope was too big in the beginning and has been reduced on all motors making the robot more stable.

When the robot begins to walk, the first action is to move all the actuators to their null positions at a slow speed with low torque. This is important so they don't damage themselves and don't hurt someone holding the robot while it is moving to its initial stance. In the initial stance all actuators are set to positions such that the robot is standing straight upwards with its arms at its waist. After the actuators reached their null positions the torque and speed are both set to the maximum so the robot is ready to walk.



(a) Draft of the motor connections, IDs and turning directions



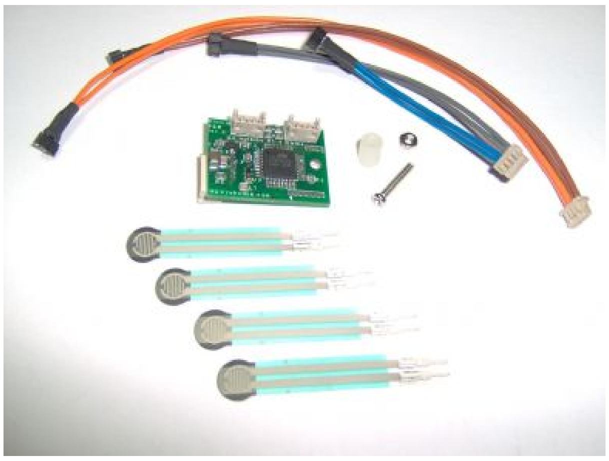
(b) Internal transition

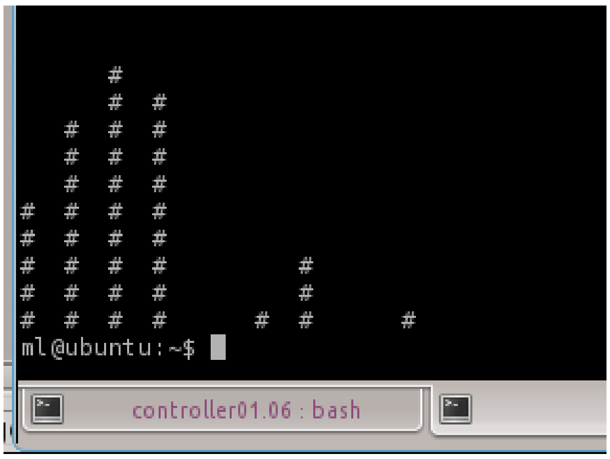
Figure 2.4.: The figures show the motor's interconnection.

Physically the motors with the numbers from 0 to 11 are connected to one USB2Serial port and the motors 12 to 23 are connected to the other USB2Serial port. Internally they are organized to two bunches of six motors on each USB2Serial port. Although most motors are connected in a series as shown in figure 2.4(a) internally they are interconnect to form a parallel circuit 2.4(b).

## 2.3. Foot sensors

HUV Robotics [foo] from Canada supplies foot pressure sensor boards equipped with MCUs. Two of those boards have been ordered and mounted on the Cycloid II's feet. Each board is connected to four force sensing resistors which have a sensing area diameter of 5mm. The detected pressure ranges from $<$1.5 psi to $>$150 psi and likewise the detected force range ranges from $<$1N to $>$100N. The boards read the analog sensor values and convert them into a 10bit digital value ranging from 0 to 1023 which is refreshed at a frequency of 100Hz. The communication protocol is similar to the motors' communication protocol [Tik] but they use TTL instead of RS485 and thereby need to have their own Port on the Cycloid II.


(a) The Foot Pressure Sensor Board kit from HUV Robotics [foo]

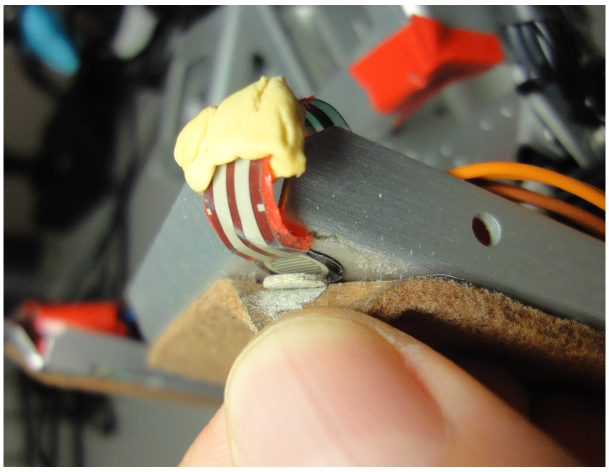

```
      #
      #  #
   #  #  #
   #  #  #
   #  #  #
#  #  #  #
#  #  #  #
#  #  #  #         #
#  #  #  #         #
#  #  #  #      #  #      #
ml@ubuntu:~$
```

(b) Foot-sensor test program printout

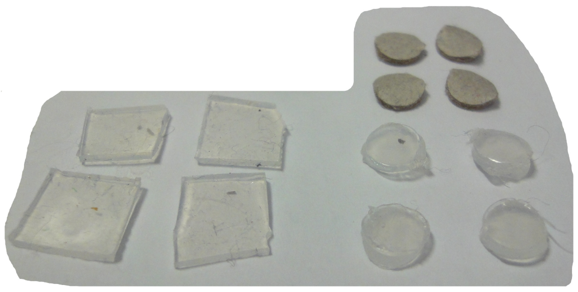
Figure 2.5.: The foot-sensor parts 2.5(a) and an example screenshot of the foot-sensor test program 2.5(b)

After the foot-sensor boards 2.5(a) were attached to the legs, a way to test their functionality was missing. Therefore a small program 2.5(b) was written which requests and reads their values at a rate of 100Hz. The program uses the command line to print out each sensor pads value by a vertical bar consisting of '#' chars where the height of the bar determines the pressure on the corresponding sensor pad. Thereby responsiveness of each sensor can be tested and a malfunction is detected quickly. This has proven to be quite useful when conducting the tests with the robot because unfortunately the sensor pads are very likely to break on their junction line when they undergo too much rubbing on it.

Another issue was the question how to best attach the sensor pads on the bottom of the foot. The foot sensor pads are only 0.5mm thick and different variants have been tested to figure out how to mount the foot sensor pads on the bottom of the foot properly as simply attaching them results in unreliable responses from the sensors. As the foot is flat, they need a little elevation under them to get the pressure to the spots where they are attached at. A layer of felt on the sole of the foot reduces the risk to slip and protects the sensor pads during usage. To improve



(a) An elevation pad between the felt and the sensor pad.

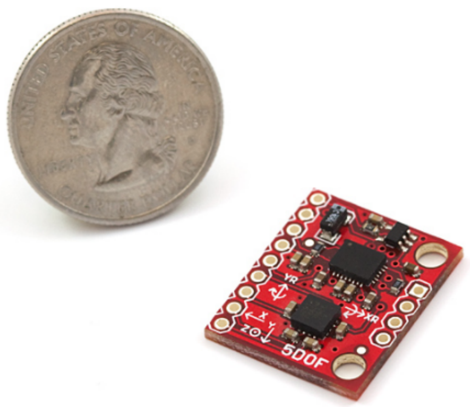


(b) Elevation pads: Silicone on the left, hard plastic on the right and cardboard above

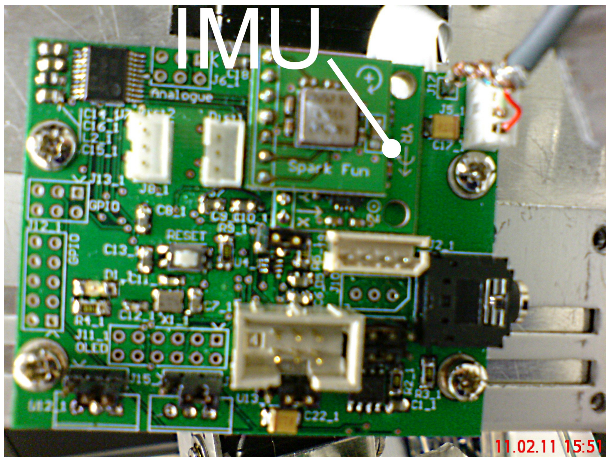
Figure 2.6.: Elevation pads and their mounting.

the sensor readings some material was put in between the sensor pad, which sticks directly on the metal sole, and the felt to get the elevation needed which is shown in figure 2.6(a). Three different materials were tested: Silicone, hard plastic and cardboard, shown in figure 2.6(b). Silicone worked very well but on the other hand it was unreliable as it does not glue or stick to anything and could fall out any time, which is bad when it isn't noticed as this causes the robot to suddenly behave differently. Hard plastic has also been tried but tests showed that it was too hard and thereby it was pressing on the sensor pads irregularly which resulted in unbalanced sensor readings. A thin layer of cardboard proved to work best as it easily sticks on the felt by the use of some glue and equally distributes the force over the sensor pads.

## 2.4. Inertial measurement unit (IMU)



(a) The IMU Analog Combo Board from Sparkfun Electronics [spf].



(b) The HySense Lite Module with the IMU mounted on it.

Figure 2.7.: The Sparkfun IMU Board mounted on the HySense Lite Module

The IMU is mounted onto the HySense Lite module 2.7(b) which is attached to the chest-board of the Cycloid II. The IMU itself is made by SparkFun Electronics from Denver [spf]. Its accelerometers measure accelerations in a range of $\pm 3$g. The gyroscopes support rotations of up to $\pm 500°$/sec. All six sensors are connected to the HySense module and therefore their values have to be requested from the module. The gyroscopes are over 10cm distant from the center-of-mass. Furthermore the manual for the HySense Lite module has not been completed so unfortunately reading the gyroscope values does not work properly. That's why they are not being used for a Kalman-fusioned calculation of the angle of the robot.

Therefore in cases where the IMU is used to calculate the angle of the robot, it is calculated using the accelerometers only. To calculate the forwards and backwards lean, the x-axis and z-axis acclerometer values are processed in the two-argument function atan2 [Wika]. This returns the angle of the IMU in radians, but as the HySense Lite module is tilted in the robots upwards standing stance, the value has an offset assigned. Furthermore the angle is filtered by merging the new values with the old ones giving the new ones a smaller weight. This doesn't delay the value much as the new values come in with a rate of 100Hz.

# 3. Theoretical principles [1]

## 3.1. Pendulum model

The Cycloid II is modeled as an inverted pendulum where the base is at the center-of-pressure along the support foot. As the feet are flat, the position of the pivot is easily controlled by movements of the ankle joint. For simplicity the base of the pendulum is discretized to be in one of three positions, at the toe, the center or the heel of the foot as shown in figure 3.1.

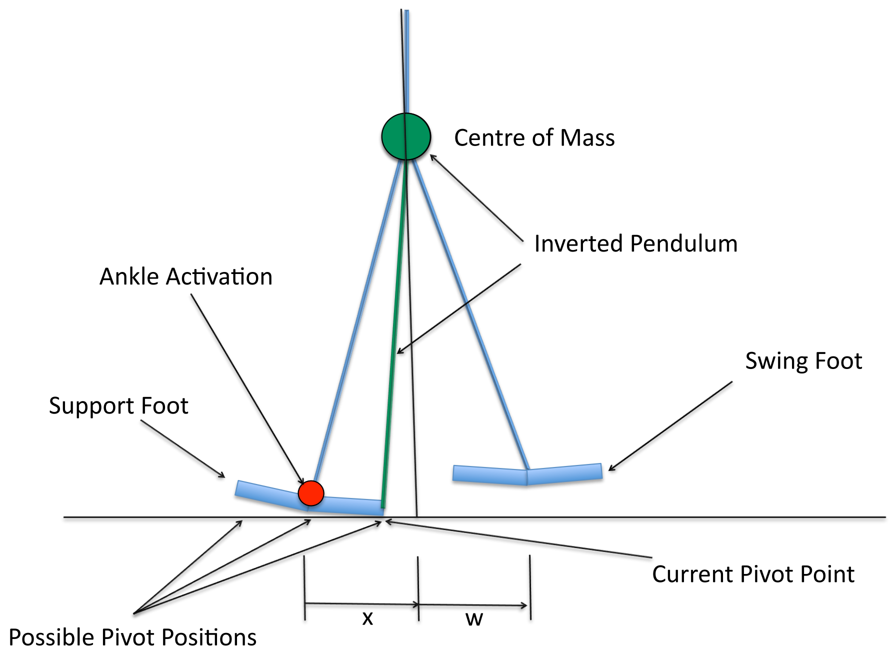


Figure 3.1.: Side-view of the robot modeled as an inverted pendulum. [HLW11]

Four variables are used to define the state $s$ of the system $(x,\dot{x},w,t)$. x is used to describe the horizontal displacement from the center of the support foot to the center of mass. $\dot{x}$ describes the horizontal velocity of the center of mass in relation to the ground. The horizontal displacement from the center of mass to the center of the swing foot is described by $w$ and $t$ describes in which time-step of the walk-cycle the system is.

A tuple $(c, d)$ defines the control actions $a$. $c$ is used to choose the center of pressure for the support foot relative to the center of the foot and $d$ is used to choose a delta change in

[1]The work on the theoretical principles including the programming of the simulator and the reinforcement learner was mainly done by Bernhard Hengst and therefore they are close to the versions in our joint paper [HLW11]. It is still included in here as it is essential for the understanding of this work.

the displacement $w$ of the swing foot making it possible to move the swing foot into position progressively.

To determine the state transition function, the progression of time, the inverted pendulum dynamics, the change in $w$ based on action $d$ and the walking gait which determines when the swing and support feet alternate are used. The system difference equations with time-step $\Delta\, t$ and indexed by $k$ are:

$$
\begin{aligned}
x_{k+1} &= x_k + \dot{x}_k\, \Delta t + \ddot{x}_k \frac{\Delta t^2}{2} && (3.1)\\
\dot{x}_{k+1} &= \dot{x}_k + \ddot{x}_k \Delta t && (3.2)\\
\ddot{x}_{k+1} &= g\, sin(\theta_{k+1})\, cos(\theta_{k+1}) && (3.3)\\
w_{k+1} &= w_k + d && (3.4)\\
t_{k+1} &= t_k + \Delta t && (3.5)
\end{aligned}
$$

In these formulas $\theta_k$ is the clock-wise lean of the inverted pendulum and $g$ is the gravitational acceleration. The angle $\theta$ depends on the base of the pendulum model $p_k$ which is determined by $c$ and it depends on the height of the center-of-mass. In reality the height of the pendulum's center of mass depends on gait characteristics of the robot, for simplicity reasons a linear inverted pendulum with a constant center of mass is used in the model whereby the angle can be calculated as $\theta_k = \tan^{-1}((x_k - p_k)/h)$.

$T$ is the period of a complete walk cycle and the double support phase is assumed to be small, hence it is ignored for the purposes of system identification. When the time passes through $t = T/2$ and $t = T = 0$ the support and swing feet alternate. When this happens, the following equations augment the upper ones to keep the state correct:

$$
\begin{aligned}
x_{k+1} &= -w_{k+1} && (3.6)\\
w_{k+1} &= -x_{k+1} && (3.7)
\end{aligned}
$$

After each walk-cycle the state-space wraps around itself which is if $t_{k+1} \geq T$ then $t_{k+1} = t_{k+1} - T$. Several frames from an animation of the inverted pendulum are shown in figure 3.2.

### 3.1.1. The reinforcement learner

The formalism used by the reinforcement learner is underpinned by a Markov Decision Problem *(S, A, T, R)*. S is a set which represents the system states and A represents a set of actions. T is a stochastic transition function $T : S \times A \times S \to [0,1]$ and R is a stochastic real-valued reward function with $R : S \times A \times R \to [0,1]$. For each time-step $k$, the system transitions from the current state s $\in$ S to a next state s' $\in$ S, given an action a $\in$ A. Therefor it receives a reward r $\in$ R. A sequence of states, actions and rewards looks like: $s_k, a_k, r_k, s_{k+1}, a_{k+1}, r_{k+1}, s_{k+2}, \ldots$. The future discounted sum of rewards represented by $\sum_{t=0}^{\infty} E[\gamma^t r]$ with $t = 0$ being the current time-step, the expectation operator E, the reward $r \in R$ and a discount rate $\gamma$ is maximized. Q-Learning is used with an off-policy temporal difference approach to learning the Q action-value function $Q : S \times A \to R$. First an optimal Q function is learned to determine the optimal control action $a^*$ for state $s$ which is $max_a Q(s,a)$.

An instance of the above simulator is now represented as an MDP for reinforcement learning. The specific values used are: T = 480 milliseconds for the period of a complete walk cycle and $h = 260mm$ for the height of the center of mass. Straightforward discretisation of the above

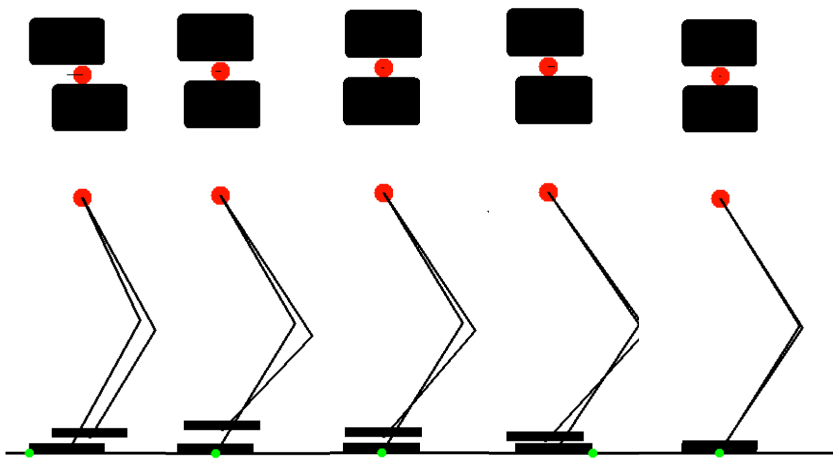


Figure 3.2.: Side view of an animation of the inverted pendulum. [HLW11] The red dot is the center of mass, the little green dot is the currently actuated base of the pendulum and the two black rectangles are the feet that are shown in plan and elevation view. The legs are not in the model, they are just drawn in the simulator for a better illustration.

| Variable | Values | Range | Increment |
|---|---|---|---|
| $x$ | 21 | -50 to 50mm | 5mm |
| $\dot{x}$ | 25 | -300 to 300mm/sec | 25mm/sec |
| $w$ | 21 | -50 to 50mm | 5mm |
| $t$ | 24 | 0 to 479 millisec | 20millisec |
| $c$ | 3 | -40 to 40mm | 40mm |
| $d$ | 3 | -5 to 5mm | 5mm |

Table 3.1.: The number of discrete values for state and action variables that are used to define the size of the Q table and their meaning in terms of a specific instance of the robot [HLW11].

continuous variables is used as linear function approximator [SB98]. The simulator's time-step takes one millisecond (1000Hz) whereas the learner runs with the same speed as the real robot which is limited by the motors and sensors to 100Hz, resulting in 10ms of processing time each step. The number of values used for the Q action-value table and the range of the variables are listed in table 3.1.

As usual in reinforcement learning, positive rewards are given for achieving certain goal states and negative rewards are chosen to avoid other states. If any of the state variables $x$, $\dot{x}$ and $w$ move out of their range an arbitrary reward of -1000 is given. Whereas for each state where the variables get close to their goal, e.g. all close to zero if the goal is to balance the robot in an upright position, a reward of 1000 is given. Small negative rewards of -1 are added if the ankle motor has to be moved to cause more toe or heel pressure and medium negative rewards of -10 are given if the movement of the swing foot is influenced, to encourage parsimonious movement.

For goals where the robot is moving with a certain velocity the reward of 1000 is given at states where $x$ is close to zero, $\dot{x}$ is at the specified velocity and $t$ is near $T/4$ and $3/4T$.

Thereby the reinforcement learner is provided with two way-points in the walk-cycle, similar to [MCAZ04] selecting actions at Poincaré sections [WGC+07] where the pendulum is upright and the center of mass has the specified velocity component. This is done with a Q-learning rate of 0.05 and a discount factor of $\gamma = 0.9$.

The simulator is a deterministic system made to provide training examples for the learner, but the function approximation makes the system only approximately Markov. A stochastic transition function approximates the transition function for the discrete system. The learning regime must be handled with care as the pseudo stochastic transitions are a function of the policy and depend on a longer history of states and actions. Here a circular phenomenon occurs in which the policy is determined by the value function and the value function is dependent on the policy.

To avoid circular phenomena and learn every action properly the simulator is started repeatedly at a random point in the state-space and executes a trajectory of 100 transitions, which is about two walk cycles, using an $\varepsilon - greedy$ exploration function that executes the latest best policy in 80% of the time. Thereby the stochastic transition function is bootstrapped to operate with the latest best policy. Using an eligibility trace that has the added advantage of speeding up learning improves the Markov approximation further by accelerating the back-propagation of the reward signal.

The reinforcement learning continues in simulation until the average reward per time-step settles to a maximum value. A typical learning profile is shown in figure 3.3. The final simulated policy $\pi(x, \dot{x}, t, w) = (c, d)$ is frozen prior to the transfer to the physical robot.

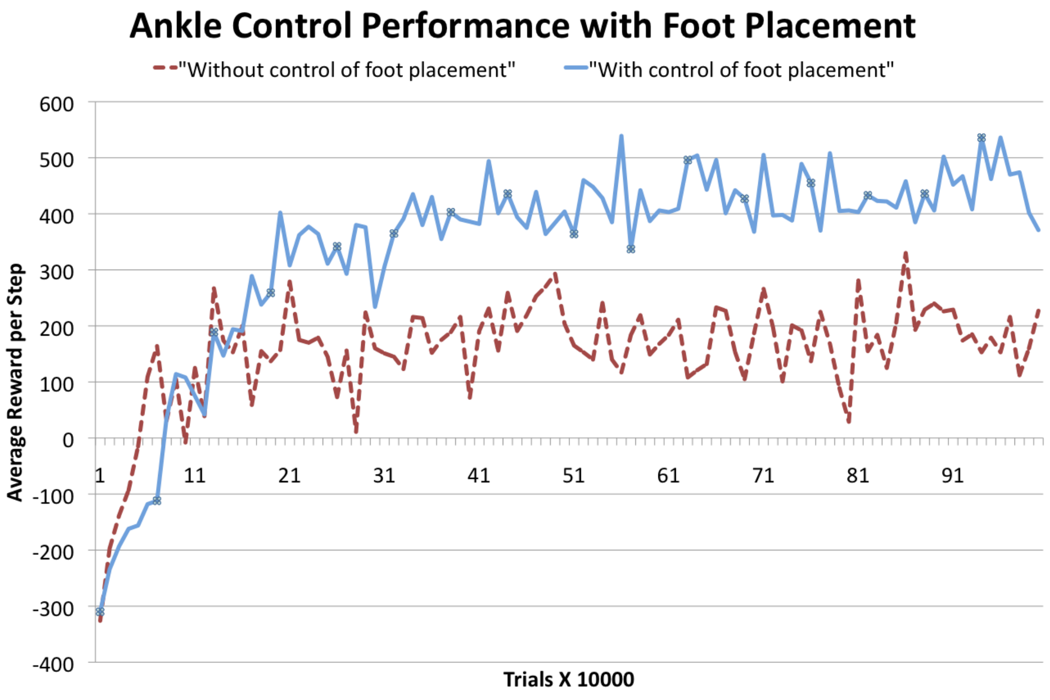


Figure 3.3.: Comparison of the Ankle Control Performance with and without Foot Placement control. [HLW11]

## 3.1.2. Simulation results

Several experiments have been conducted in the simulator to find out about the robustness of the learnt policy against impact forces and random changes in the walking direction.

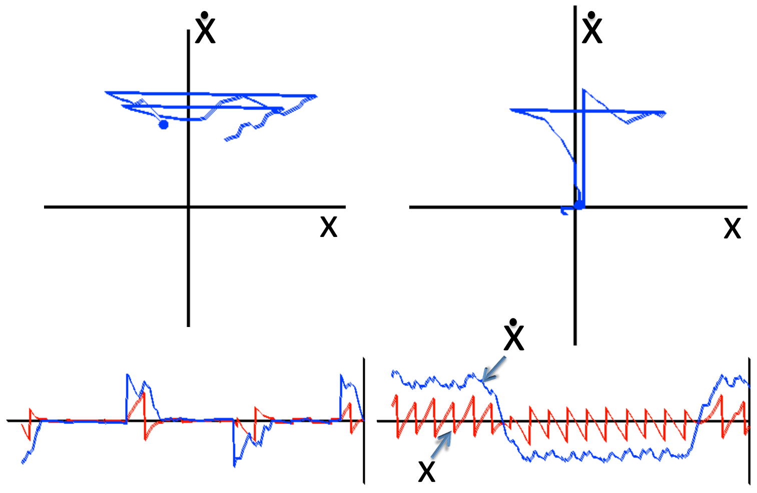


Figure 3.4.: x (red) and $\dot{x}$ (blue) plotted against the time. On the left: sudden impact forces. On the right: sudden changes in the walking direction. [HLW11]

The first set of experiments was conducted only moving the ankle to control the pivot. The simulated swing foot movement $(w)$ was set to $x$ whereby the swing foot mirrored the support foot placing the swing foot a total of $2x$ from the support foot. Figure 3.4 shows four diagrams where $x$ and $\dot{x}$ are plotted against time in two ways. The upper two diagrams show the time-trace on a coordinate system with horizontal axis x and vertical axis $\dot{x}$. The unfolding time series for x (red) and $\dot{x}$ (blue) with the current time on the right is shown by the two bottom diagrams.

The two diagrams on the left of figure 3.4 show the time-series response of x and $\dot{x}$ to induced changes in $\dot{x}$ which simulate sudden impact forces. The graph shows that the deceleration, which is induced by the actuated ankle joint to fight the impact force, persists over several walk cycles. The learner is able to plan ahead and take appropriate actions in anticipation of support foot changes due to the inclusion of the time variable $t$ as part of the state of the system.

The two diagrams on the right of figure 3.4 show sudden changes in the walking direction. As one controller is trained for a certain velocity, controllers have to be switched when changing the velocity. What can be seen in the diagram is a switch from a forward walking controller to a backwards walking one. As the controllers are learned to reach their goal as quickly as possible, the transition is fast and immediate but still smooth, even with the swing foot being in mid-stride. Furthermore the figure shows that only controlling the ankle motors directly is already enough to make the robot move robustly but still de- and accelerate quickly.

In simulations where the swing foot gets controlled directly by incremental movement actions $d$ affecting the swing foot position $w$, the swing-foot placement is done in conjunction with the ankle control to optimally arrest the motion of the robot after sudden impact forces. The typical response to bi-directional impulse forces for a 40mm width foot with foot-placement control is shown on the right of figure 3.5 whereas the response without foot-placement control is on the left. It shows that ankle control alone takes longer and several steps to balance the robot for a robot with small feet.

As the foot-placement control makes the system more complex, simulations have been conducted to find out how beneficial it actually is. The average reward per step with foot placement control enabled and disabled is shown in figure 3.6. Unsurprisingly the increase in foot size wipes out the difference.

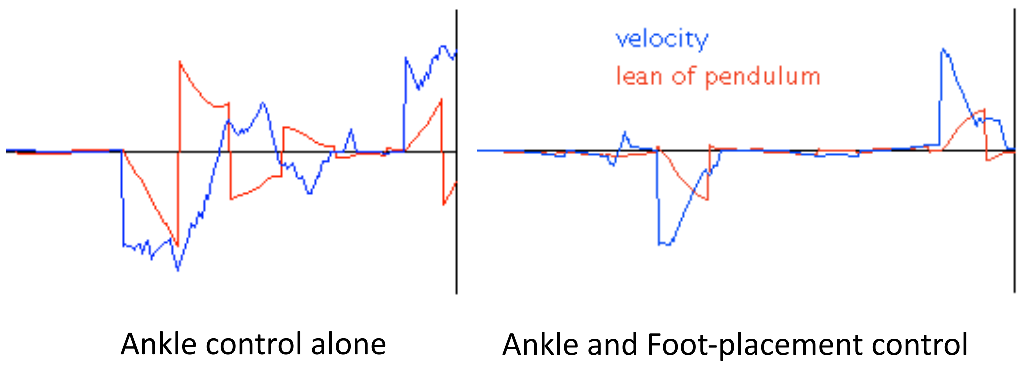


Figure 3.5.: Time it takes to get back to balance, with and without foot-placement control for a small 40mm foot. [HLW11]

The crude three-value discretization for the pivot position is the reason for the drop of the average reward per step in both scenarios. The simulated ankle control torque at the heel and toe gets bigger with larger feet making it difficult for the learner to keep the robot in the part of the state-space that achieves the high reward. As the foot size of the Cycloid II is 106mm, figure 3.6 shows that direct foot-placement control would not be of much benefit, thus the focus lies on ankle-tilt control in the physical experiments.

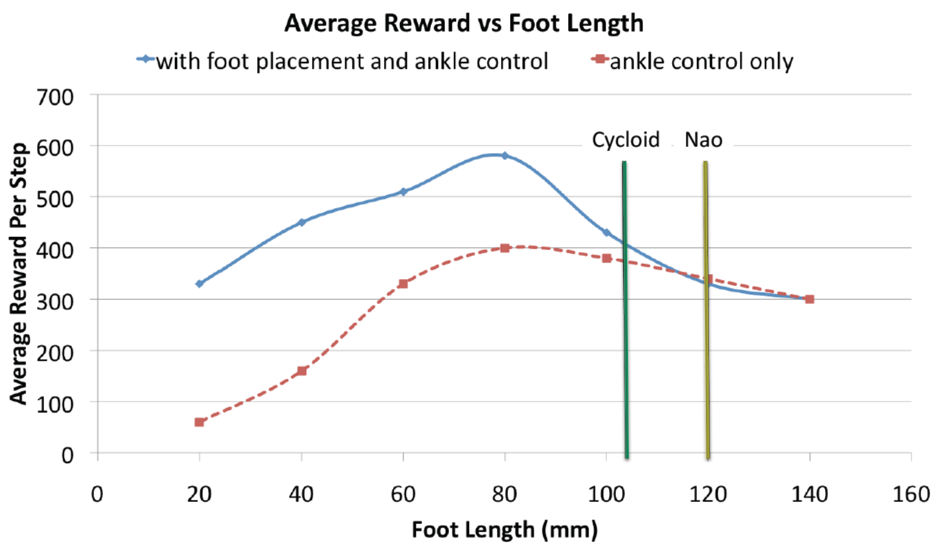


Figure 3.6.: Comparison of the average reward received with and without foot-placement control in relation to the length of the foot. [HLW11]

The learner still takes the swing-foot policy into consideration when deciding about the ankle control policy, even with the swing foot being placed in relation to the CoM and the support foot. The policy for $x$ and $\dot{x}$ is shown in relief, half-way through a swing phase (at t = 360ms) is shown in figure 3.7.

Noticeable is the thin green/red line going through the middle and getting bigger in the second and fourth quadrant. In these regions little to no control has to be performed as the robot either is in balance or the position of the support foot and the acceleration are about to compensate each other some steps later resulting in balance. More controlling has to be done in the opposite cases as seen in the first and third quadrant where the lean of the pendulum and the current acceleration are not about to compensate but about to enhance the acceleration which could lead to a loss of balance. Interesting were the islands in the the middle of the first quadrant as well as in the middle of the third quadrant. I suspect they were created becauase in this part of the state space it was better to wait for the next change in support foot than to arrest the velocity. Little control is done at the extreme which is in the upper right corner of the first quadrant as well as in the lower left corner of the third quadrant which is probably the case because there was no more chance of getting the robot back to balance anymore. In both instances, the island areas and the extreme cases, no action was performed as performing no action does not give any additional negative reward, compared to an action that can't stop the falling (but additionally creates negative reward as every action does).

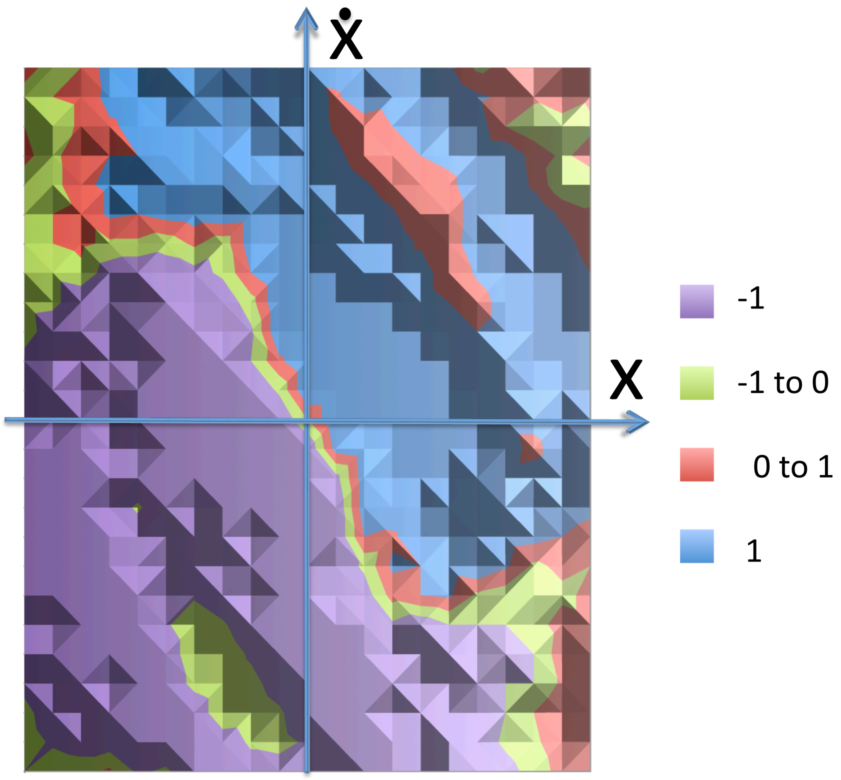


Figure 3.7.: The policy for $x$ and $\dot{x}$ in relief representation at t = 360ms.

# 4. Implementation and evaluation [1]

## 4.1. Overview

Policies which performed well in the simulator were tested on the real robot. To keep the center-of-mass at a constant height, a walking gait that drives the hip, knee and ankle motors using closed form kinematics was utilized. The support foot is always assumed to be close to flat on the ground while the robot moves, and the swing foot is kept parallel to the ground. Unlike in simulation the state of the robot is not omnipresent but has to be estimated based on sensor readings. These sensor readings come from an IMU unit with accelerometer and gyroscope, from the feet which are equipped with four pressure sensing foot-sensors per foot and from the motors which all have encoders to tell the motor position.

A recursive Bayesian filter [WB95] is used for the state estimation. To reduce the amount of on-line computation and perform recursive updates to estimate the state variables $x, \dot{x}, \ddot{x}, w, t$ a steady-state Kalman filter is utilized. Two things are done by the filter:

- First a prediction update is done using the linear inverted pendulum model equations from the simulator with the pivot point being estimated from center-of-pressure calculations which are based on foot-sensor readings.
- Second a correction update which is based on kinematic, IMU and foot-sensor observations is done with a constant gain matrix of [0.5 0.5 0.5 0 0] used to update the variables $x, \dot{x}, \ddot{x}, w, t$.

## 4.2. Motion-cycle

To get an approximation of the location of the base of the pendulum along the support foot, the foot-sensor readings are used to calculate the center-of-pressure. The origin is located directly under the ankle joint and the displacement in mm is calculated by taking the weighted average of the foot-sensor readings of the actual foot.

$$p(foot) = \frac{\sum d_i * f_i}{\sum f_i} \tag{4.1}$$

Where $foot \in L(left), R(right)$ *and* $d_i$ is the horizontal distance from the ankle joint to the corresponding foot-sensor $i$ with $f_i$ being the sensors reading.

The process update uses the CoP as the pivot point of the inverted pendulum. The walk-cycle is set to be fractioned to two swing phases surrounded by the double support phases D, the first swing phase ranges from $t = D/4$ to $t = T/2 - D/4$ and the second one from $t = T/2 + D/4$ to $t = T - D/4$.

[1] The paper [HLW11] which I contributed to with this project also describes the Physical Robot Implementation, which is described here, up to the section "Implementation of the controller" 4.3 in similar ways.

The coronal center-of-pressure component defines the support foot by determining where the base of the pendulum from the frontal perspective is located. The sidewards rocking motion of the robot is achieved by some kind of bang-bang control [SVV64]. This control switches the support-foot when the base crosses the zero-crossing point between the feet. A polygon which is stretched by the feet with the origin between them is used to measure this crossing of the zero-crossing point.

To make the rocking motion of the walk-cycle more natural the period T was adjusted until it seemed to be close to the robots natural frequency. Just taking the frequency of a pendulum with the length of the height of the robots center-of-mass would not work as the robot underlies spinning forces and other influences.

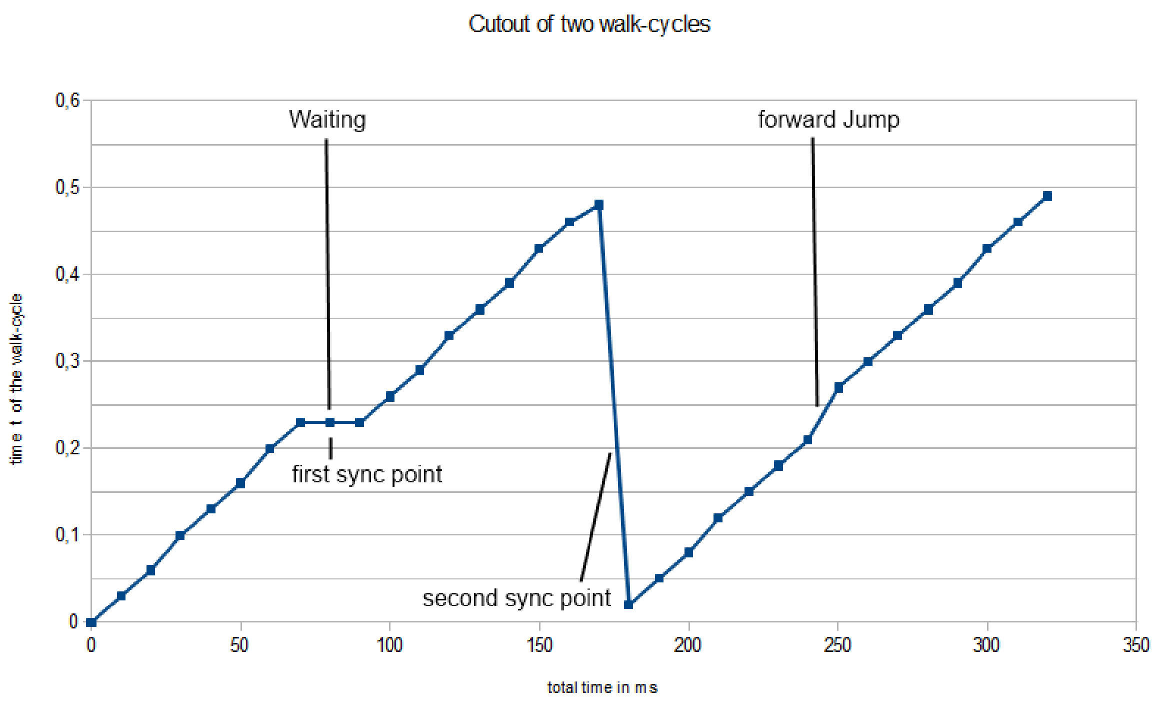


Figure 4.1.: Two walk-cycle examples which show the synchronization between the walk-cycle and the swing of the robot.

Even with the period $T$ being close to the natural frequency of the robot it often happens that the robot gets disturbed by outer influences whereby the swing leg touches the ground before or after half of the period $t = T/2$ (first sync point) or the full period $t = T$ (second sync point) is over. In those cases the rock is synchronized at these times where the support foot is supposed to switch in any case. If the leg gets on the ground earlier, then the inner time is shifted forwards (figure 4.1 right). In case the leg has moved completely up and down but doesn't touch the ground yet (figure 4.1 left), which means that the robot swung more sidewards than normally, the inner time is paused until the leg touches the ground and the sum of the pressure on the foot sensors gets higher than the sum of the pressure on the foot sensors of the other foot. This happens when the zero-crossing point has been crossed. Thereby it is ensured that the leg's motion is in sync with the robots rock.

## 4.3. Implementation of the controller

When the policies for balancing on the spot and for walking are learned properly, they are written into a file with one line for each state represented by the values of the state variables (found in table 3.1) and the corresponding action in the end of each line. This file is then read by the robot and written into a multidimensional array with one dimension for each state variable being of the size of the variable after discretization. The action for the momentary state is quickly looked up at runtime by discretizing the actual state variables and feeding them into the array giving the optimal control action determining how to set the center-of-pressure on the support foot.

The control action c is also discretized and can only be one of the three values $-1, 0, 1$. They indicate how the ankle should be controlled to move the center-of-pressure as follows: (-1) indicates that the CoP should be at the back of the foot, (0) means around the origin under the ankle joint and (1) stands for the front of the foot. How this control action is actually handled depends on the current position of the center-of-pressure. The variable $p$ indicates where the CoP currently is and is again discretized to the values of $-1, 0, 1$ with (-1) standing for the back, (0) for the middle and (1) for the front of the foot. The discretization from the CoP value to $p$ is realized by splitting up the foot in three roughly similar sized areas. The origin and everything $\pm 20mm$ from the middle of the foot is the middle area (0), everything further in the front is discretized to be at the toes (1) and everything further back is the heel area (-1). There are instances where no more effect can be achieved. This happens if the control action decides to put the pressure to the front but the pressure is already at the front or vice versa, thus tilting the ankle-pitch more in that direction wouldn't be beneficial. Also if the pressure shall be moved from the heel to the toe (or the other way around) at once, a greater movement is necessary. This has been implemented using the following equations where $\Delta f$ is the ankle-pitch movement:

```
If $D/4 \leq t < T/2 - D/4$ or $T/2 + D/4 \leq t < T - D/4$:
  if $c = -1$
    if $(p = -1)$ no-change
    if $(p = 0)$ $a = \Delta f$
    if $(p = 1)$ $a = 2\Delta f$
  if $c = 0$
    if $(p = -1)$ $a = -\Delta f$
    if $(p = 0)$ no-change
    if $(p = 1)$ $a = \Delta f$
  if $c = 1$
    if $(p = -1)$ $a = -2\Delta f$
    if $(p = 0)$ $a = -\Delta f$
    if $(p = 1)$ no-change
```

If there is already force where the control action decides it to be, then nothing is changed. But if the control action decides the pressure to be on the other end of the foot than it momentarily is, double the movement is done.

The tilt of the ankle motor by $\Delta f$ has been limited to a maximum tilt where the robot is close to falling over but does not fall over yet, based on the robot standing still in an upright position. For $\Delta f$ different approaches have been tested. The first approach used a small value for $\Delta f$ which would then be able to stack up to the limits depending on the control action.

The idea behind that approach was that the foot could change its inclination smoothly but still fast enough to adjust and adapt precisely to disturbances as the machine runs at 100Hz. After some testing, the outcome was that better results were achieved with larger $\Delta f$ and the best results were achieved when $\Delta f$ was half as big as the total tilting range of the ankle. Thereby the ankle changes from the maximum tilt in one direction to the maximum tilt in the other direction instantly in case that $2 \cdot \Delta f$ is applied. Fortunately no increase in jerky behaviour could be noticed with a bigger $\Delta f$ but great improvements in handling disturbances and better responsiveness were noticed. These improvements are explainable due to the even faster and stronger actions that are performed with a greater $\Delta f$. Those faster and stronger actions provide the controller with more influence, which is still manageable as the cycle time of 10ms is quick enough to only induce a small impulse to the system if that is all that is necessary.

## 4.4. Basic rock and walk

Whenever the robot walks forwards,backwards or on the spot it actually rocks from one foot to the other, from side to side. This rocking is realized by alternately lifting one leg up and down while the other leg stays on the ground. To move a leg vertically simple trigonometric functions are used which calculate the angles for the hip, knee and ankle actuators to the requested height. As the walk follows a fixed time period, the vertical movement follows a curve function which makes the leg go up to a set max height and back down on the ground smoothly during half of that period.

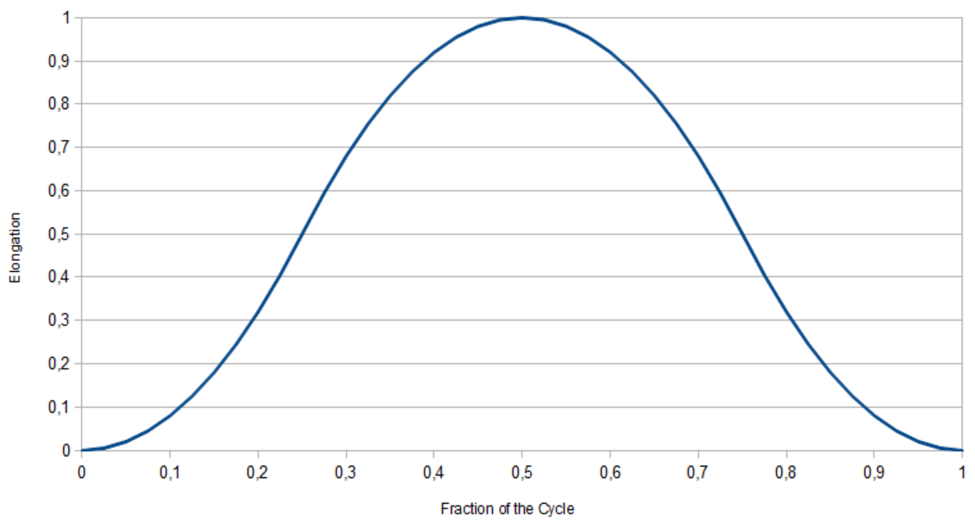


Figure 4.2.: This figure shows the horizontal movement of the leg when lifted. Following the formulas 4.2.

$$lift = \begin{cases} 8 \cdot t^2, & for \;\; 0.00 \leq t < 0.25 \\ -8 \cdot t^2 + 8 \cdot t - 1, & for \;\; 0.25 \leq t < 0.75 \\ 8 \cdot t^2 - 16 \cdot t + 8, & for \;\; 0.75 \leq t < 1.00 \end{cases}$$

(4.2)

To make the robot walk forwards and backwards another function is used which moves the legs horizontally. As for the vertical movement the angles of the hip, knee and ankle joints are observed and calculated. When the robot moves without the controller, a step size is given and the same curve function as for the vertical movement is used to make the horizontal movement smooth by first accelerating it and then decelerating the motion instead of quickly forcing it to a motion with constant speed and instantly stopping it when the step size is reached. Combining the vertical and horizontal walking functions by adding up the motor angles makes the legs walk forwards or backwards.

While the swing-leg gets lifted up and repositioned, the support-leg, which stays on the ground, moves horizontally in the inverse direction. Thereby the legs get alternately moved in one direction while in the air and in the other direction while on the ground realizing the walking motion. When the swing-leg moves horizontally, it moves in the opposite direction to the support-leg following the curve between 0 and 0.5 shown in figure 4.2. This makes the swing-leg movement smoother as it will start slow, get fast and slow down again in the end when reaching the final position where it will touch the ground again. Additionally the swing-leg only moves 90% of the $x$ value calculated for the pendulum to make up some energy loss in the physical machine. Without this decrease the robot ended up always stepping a bit too far whereby the pendulum ended up falling down ”between the robots feet” while the robot was doing the splits.

## 4.5. Pendulum controller

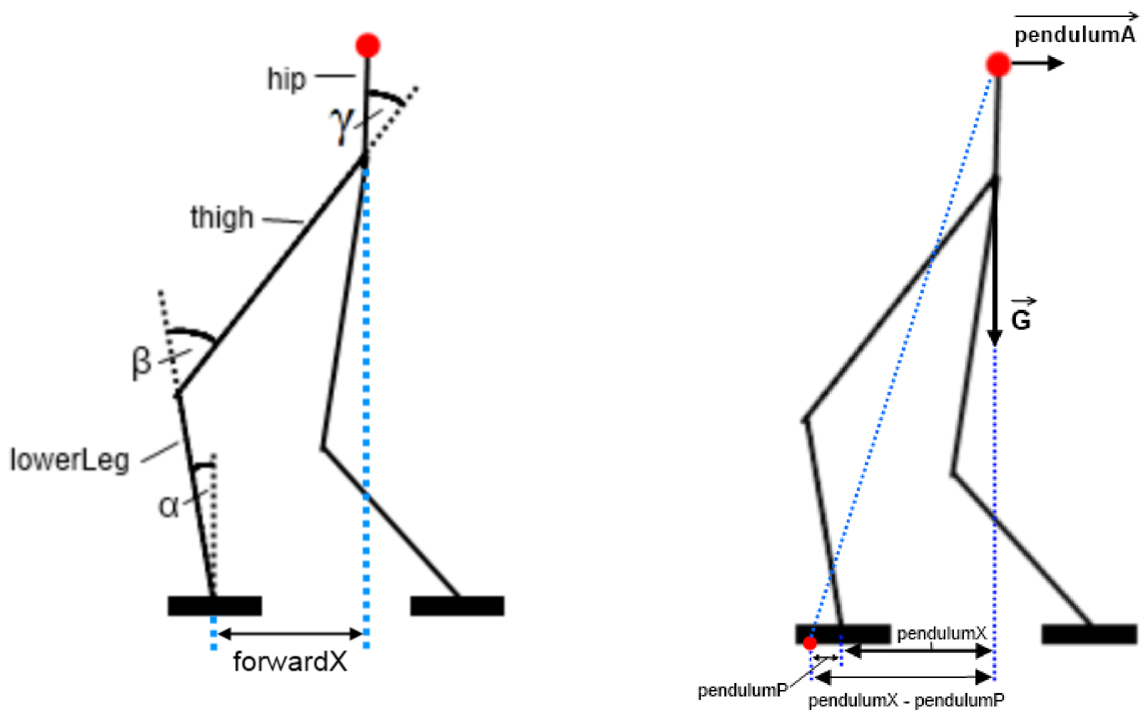
When the robot rocks and walks via the Pendulum Controller it uses the same functions as described in the previous section for the vertical movement of the foot. Whereas the step width is controlled by the pendulum $x$ instead of by a curve function proceeding it up to a certain step size. To keep the estimation of $x$ as accurate to the real $x$ as possible it is observed and corrected with a Kalman filter and then predicted for the next step.

For the observation of $x$, the motor positions which are read every cycle are used to calculate the horizontal distance between the center of each foot and the center of mass of the robot using simple trigonometry 4.3 assuming that the feet are parallel to the ground. The steady-state Kalman filter uses an $\alpha$ gain of 0.5 and a $\beta$ gain of 0.2 to keep $x$ and the velocity noise-free and up to date. The corrected values of $x$ and $\dot{x}$ are then predicted using the foot-sensors and the IMU data.

First the acceleration of the pendulum is predicted by calculating the lean of the pendulum using the foot-sensors to calculate the center of pressure on the foot which equals the position of the pendulums base. By knowing the position of the base and the center of mass the displacement between them can be calculated and thereby the acceleration can be determined using the gravity, which is shown in the following formulas 4.4.

As the sinus equals the value of the radian measure for small angles, its calculation is omitted. The distance between the pendulum base and the center-of-mass got a constant value assigned (comHeight) as its variation is so little compared to the change in the horizontal distance between the pendulum base and the CoM (pendulumX - pendulumP) that it is negligible. An illustration is given in figure 4.3(b).

$$forwardX = \sin(\alpha+\beta+\gamma)\cdot hipLength + \sin(\alpha+\beta)\cdot thighLength + \sin(\alpha)\cdot lowerLegLength \quad (4.3)$$



(a) forwardX Illustration (b) pendulumA Illustration

Figure 4.3.: Visualisation of the trigonometric calculation of the forwardX value via trigonometric formula 4.3.

$$pendulumA = GRAVITY\cdot(pendulumX - pendulumP)/comHeight;$$
$$pendulumA = pendulumA\cdot 0.5 + 0.5\cdot(IMUAccelerationAngle\cdot GRAVITY); \quad (4.4)$$

Additionally the angle determined by the IMU (only using the accelerometers) is used to calculate the acceleration of the pendulum via the gravitational constant. Both of these acceleration values are merged to one value by giving each of them half the weight. Thereby the value is more reliable, still peaks when the robot steps on something or gets pushed but is also protected to an overreaction by one of the two sensors through the redundancy.

PendulumX always equals the forwardX value of the support-foot and all signs are in respect to the implementation.

The acceleration then gets multiplied by 10ms which is the time of a cycle to calculate and add the change in velocity. This new velocity then gets added to the $x$ value used as a prediction for the next state. This prediction is used to calculate the next positions sent to the motors.

An important situation is when the support- and swing-leg change because the observation value suddenly jumps. To prevent the new swing leg from making a hectic jump, each time the legs change, the difference between the new and old value is taken and added to the swing leg position. The amount which is added up to the swing leg is constantly reduced during the phase until the legs change again.

## 4.6. Threaded read/write using mutex locks

To be able to do some calculations while communicating with the sensors, the system is running two parallel threads as shown in 4.4. The control-thread is mainly used for the calculations while the request-thread concurrently communicates with the HySense Lite Module which provides the accelerometer and gyroscope values from the IMU. Mutex locks are used to keep the two threads synchronized.
The control-thread starts first and posts the request-thread lock before it starts its calculations. The request-thread then sends multiple packets to the HySense module to request the sensor values. After each request packet the thread sleeps for 1.5ms to allow some time for the answer packets to arrive before the next request is sent. This prevents an overlapping of the answer sent by the HySense module and the following request.
When the control-thread is finished with its calculations, it will request the values from the walk-motors and from the foot sensors. No sleeping time is needed as the foot sensors and the motors are connected to different ports as well as all the requests to the foot sensors and to the motors are sent out at once and the answers are timed by the return delay time 2.2.

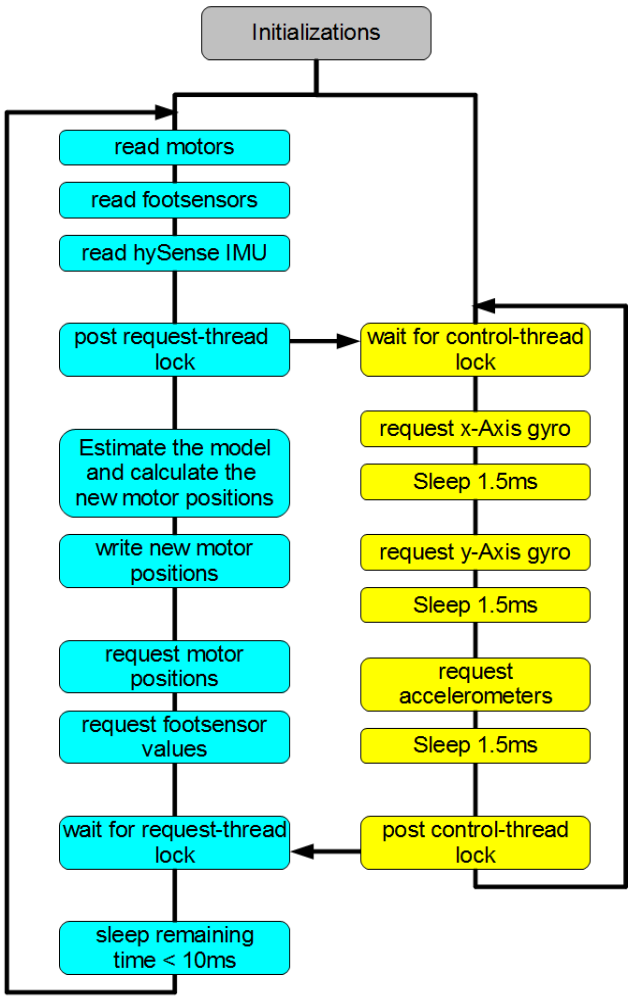


Figure 4.4.: Concurrency between the Control-thread (blue) and the Request-thread (yellow).

After both threads are finished with their calculations and sent their commands/requests, the control-thread will sleep for the remainder of the 10 ms cycle time to ensure that the system runs at 100 Hz. After that it starts its loop again which begins with the reading of all the motor and sensor values which were requested and arrived in the buffers in the meantime. After that the request-thread lock is posted to let the request-thread run again and the next calculations are started.

If the positions of more than the six walk-motors should be read, another set of request packets would have to be sent out to up to another six motors. Some milliseconds of waiting/sleeping time would also be needed between those two requests to allow the answers of the first request some time to arrive and prevent an overlapping with the next request packet. It is best to send the new commands out to the motors as soon as they have been calculated. This sending also has to be taken into account for the decision, when the position of additional motors should be requested. As these packets must also not overlap with the requests and with the answers from the motors.

Thus the best solution would probably be to create another thread which is dedicated to the communication with the motors and synchronize it with the control-thread. The problem would be to decide in which order to send out the commands as it would be better to first send the new positions to the motors before requesting their positions again. But if the calculations would take a couple of ms, this duration would have to be used to send out the new requests to the motors as this also takes a couple of milliseconds. The disadvantage would then be that the next used motor positions were requested from the motors, even before they got their new positions assigned. Thereby the calculations would be done with older values. But as the walking-system doesn't need much calculations, this problem would not come up in the near future and might probably be solved by a higher prioritisation of the locomotion threads.

## 4.7. Evaluation by tracking the X

The closer the estimated state of the machine is to the reality, the more likely it is that the robot will perform the best actions in each situation. To find out how accurate the inner state of the robot complies with the reality, a small program has been written to investigate this as used in [HLW11].

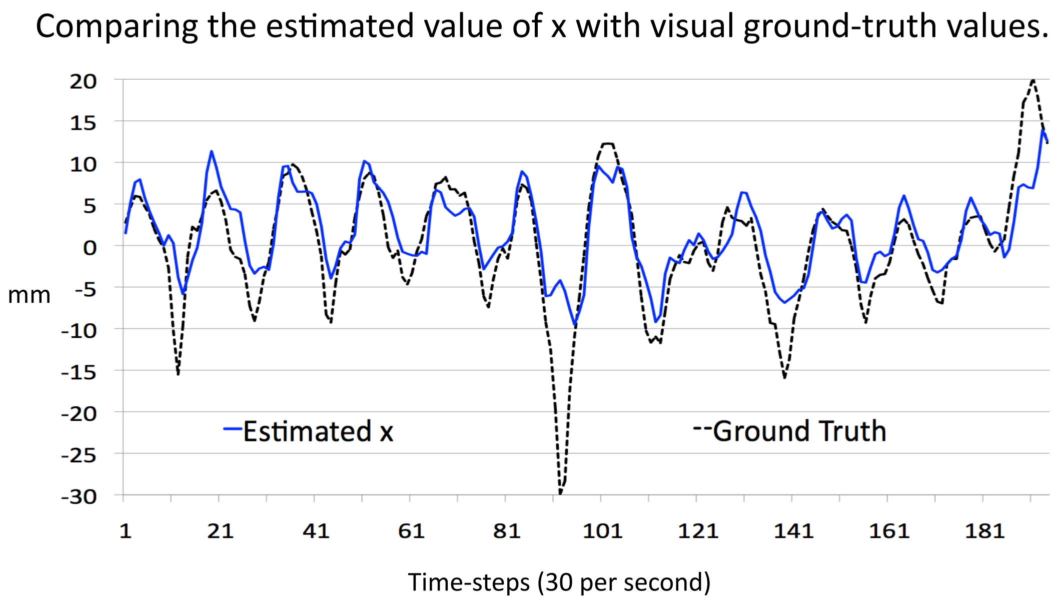


Figure 4.5.: Checking the compliance between the state estimated by the machine and the visually observed values. [HLW11]

The program uses a webcam to track two dots on the robot by calculating the highest occurrence of a certain colour through the horizontal as well as through the vertical lines of each frame. A blue dot is attached on the side of one foot and a green dot is attached at the center of mass, which is visible in 4.6. With the camera facing the robot sidewards it can track the two dots and thereby calculate the horizontal distance between them. This horizontal distance equals the inner state of the pendulumX value. While the camera runs at 30 frames per second the logging of the robot has been set to match that frequency to be able to easily compare both log files to each other.

The diagram 4.5 shows the ground truth values obtained via the camera and the estimated x value from the robot. A good relation between them is clearly recognizable. Some spikes can be found in the ground truth values that are not visible in the internal robot state which is likely to be caused by the filters. While tracking the values, the robot was mainly rocking on the spot and has been slightly disturbed by light pushing which can be seen in the middle of the diagram. Overall the compliance was absolutely satisfying.

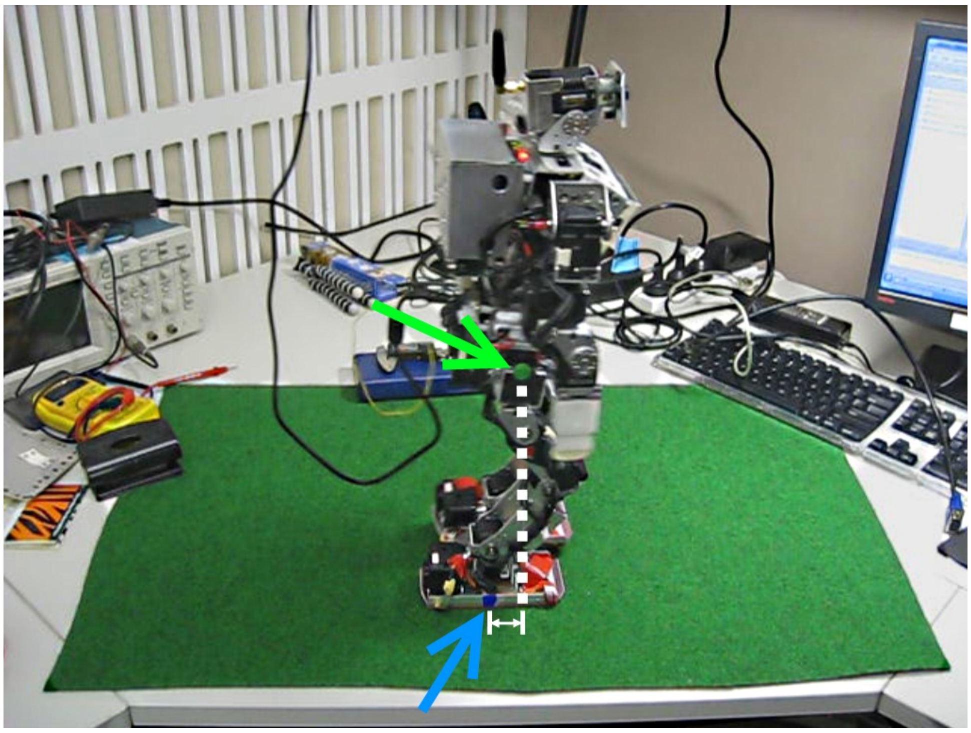


Figure 4.6.: The robot shown from the side, with a blue dot attached to its foot and a green one on its hip at the COM. The horizontal distance between them, which is measured, is indicated by the white lines and the white arrow between them.[HLW11]

# 5. Experimental results

## 5.1. Overview

Aside from the theoretical aspect it is important that things also work in practice to prove that the ideas are realizable. Different experiments have been thought of to show the abilities the robot gained through the use of the pendulum controller compared to the robot acting without the pendulum controller. Three scenarios have been found which are a challenge for any walking robot and show the improvement by the use of the pendulum controller. All of them were conducted on a grass-felt surface which we chose because the robot doesn't slide on it.

Another test which was thought of first, was called the "Bottle Test". Here the robot was supposed to walk on the spot and get hit by a plastic bottle on a thread without falling over. The thread was fixed at a bar above the robot, whereby the bottle would function as a pendulum which hits the robot at its lowest point with the power of the push being determined by the displacement of the bottle. Thus the robot could have been tested and compared with and without the pendulum controller enabled. Unfortunately the result was very dependent from the position where the robot was hit and even more so at what time of the walk-cycle it was hit. As it was not possible to conduct and time the test that precisely, it failed. The diagram 5.1 shows the inner state when the robot got hit on the front and fell backwards.

Whenever something is written about the phase in the following tests, it means the internal state which defines which leg is the support-leg and the swing-leg. The phase changes when the legs change their state from support-leg to swing-leg and vice versa.

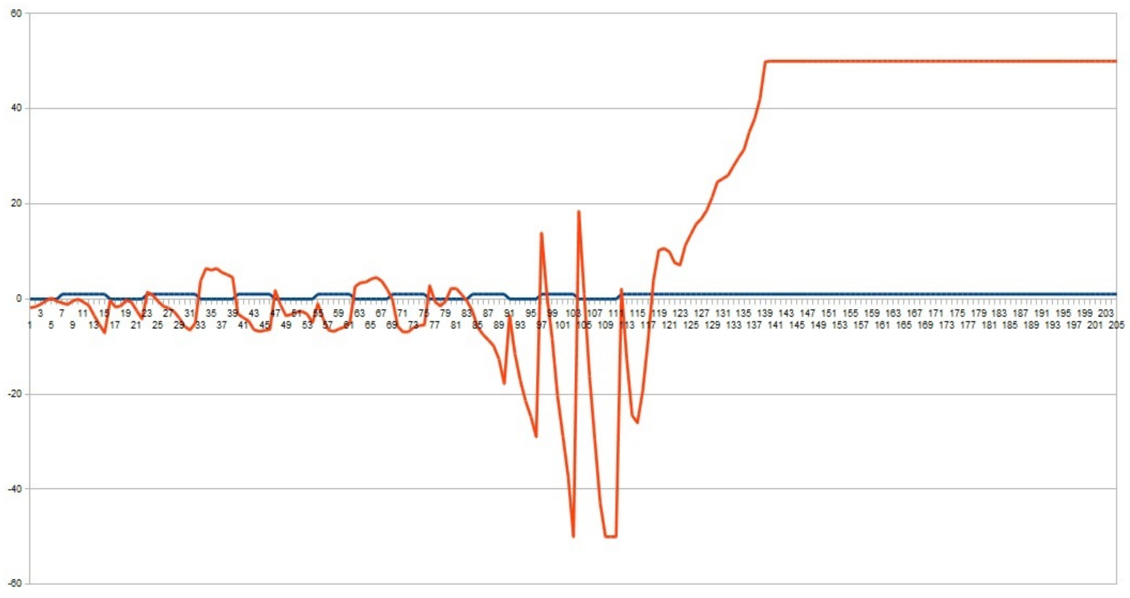


Figure 5.1.: The internal pendulumX value, shown in red, raises to the limit after the robot failed to keep the balance and the phase, which is shown in blue, stops to change.

## 5.2. Quick policy change

The first experiment demonstrates the flexibility of the robot by instantaneously changing the movement direction. At first the robot is told to move forwards and a counter is started which repeats to reverse the robots direction after some time elapsed. What actually happens is that the goal of the pendulum controller gets swapped from forwards to backwards and vice versa. The effect of this change in the program is the reading of another position of the array which contains the controller actions. Thereby the robot will automatically optimally accelerate in the opposite direction until it reaches the velocity learned for that action without the need of any further calculations.

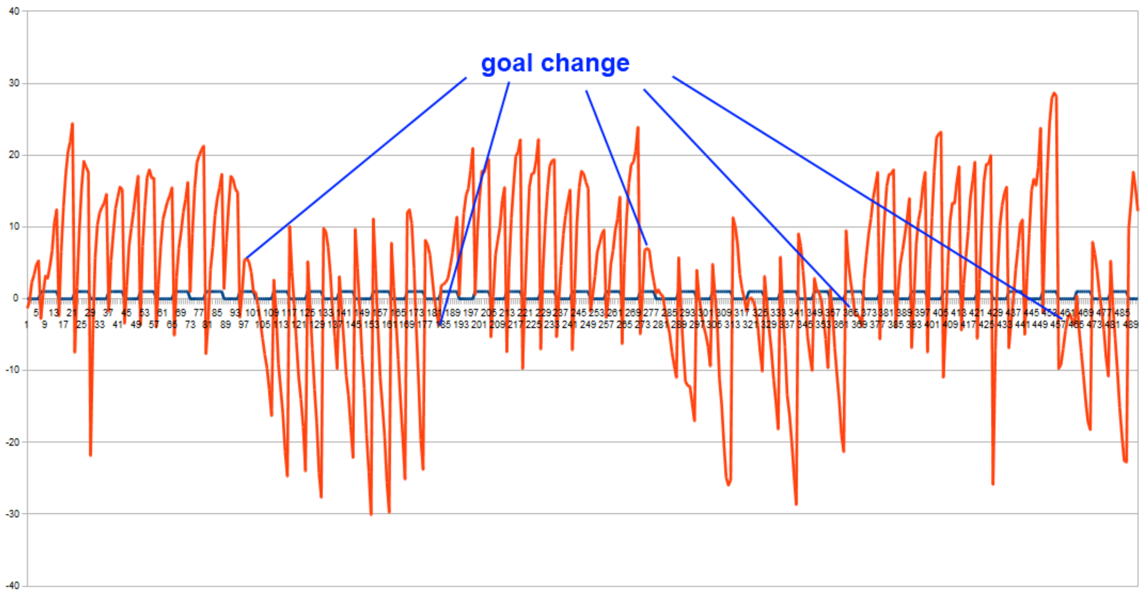


Figure 5.2.: Multiple changes in the walking direction are plotted. The red line is the pendulumX value which shows five regions that are alternating walking directions, starting forwards. The blue line is the phase again.

The diagram 5.2 shows the systems phase in blue and the pendulumX in red. Whenever the phase is high the left foot is the swing foot and the right foot is the support foot and vice versa when the phase is low. Clearly noticeable are the five areas in which the pendulumX value is on average below or above the zero line. During the time where the pendulumX's average value is positive the robot is walking forwards as a positive pendulumX means that the CoM is in front of the pendulums base and inside the areas where the pendulumX's average is negative the robot walks backwards as the CoM is behind the pendulums base.

## 5.3. Stepping test

### 5.3.1. Stepping experiment

The second experiment conducted with the robot shows the improvement in stability by the use of the pendulum controller. An external force is induced by making the robot step on something. This is first detected by the foot sensors as they will notice a much higher force on the foot at the position where it steps on the obstacle. As soon as the disturbance causes the robot to begin to lean or fall the IMU will also notice the change of the angle of the robot. As both sensors contribute to the state estimation, the system will notice the disturbance quickly and react to it. The intention of this test is to show that the robot keeps stable even with disturbances, it is not the aim to make it walk onto steps.

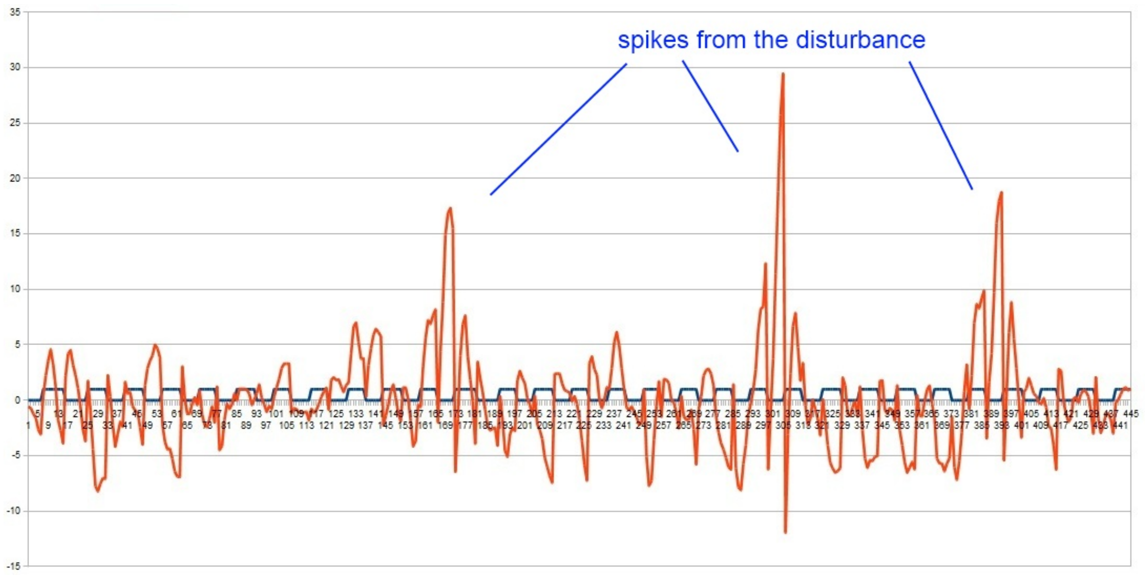


Figure 5.3.: Three spikes are visible which indicate the disturbance brought into the system by stepping on an obstacle.

The diagram 5.3 shows the systems phase in blue and the pendulumX in red. Whenever the phase is high the left foot is the swing foot and the right foot is the support foot and vice versa when the phase is low. Remarkable are the three areas where the pendulumX value spikes drastically. This indicates the robot stepping on the obstacle and its reaction to handle it. The back of the foot stepped on the obstacle which lead to a forward lean and thereby to a forwards acceleration. To manage this the robot takes a few small steps forwards and thereby gets balanced and stable again.

## 5.3.2. Stepping comparison with and without the pendulum controller

To prove that the controller is an improvement over the normal walk, robots with both controlling mechanisms had to slowly walk on an obstacle with different heights and show which heights they could handle without falling over. A book was a good obstacle as the height could be varied fast and in tiny steps by browsing the pages. A caliper was used to measure the height of the amount of pages the robot was able to step on. For each height the robot had ten attempts to try to manage it. As the normal walk didn't react to disturbances the robot kept trying to walk forwards and either got onto the obstacle or faied and fell backwards.

The blue line in the diagram 5.4 is the robot's performance when it is walking normally, whereas the red line shows the robots performance with the pendulum controller activated. The x-Axis is the height of the obstacle in mm and the y-Axis shows the success rate during the ten attempts tested for each height.

An extra 70% of step height are reliably survived by the use of the pendulum controller. It is also interesting to note that the walk using the pendulum controller could manage higher obstacles when the experiment was conducted on a table with a slippery plastic surface instead

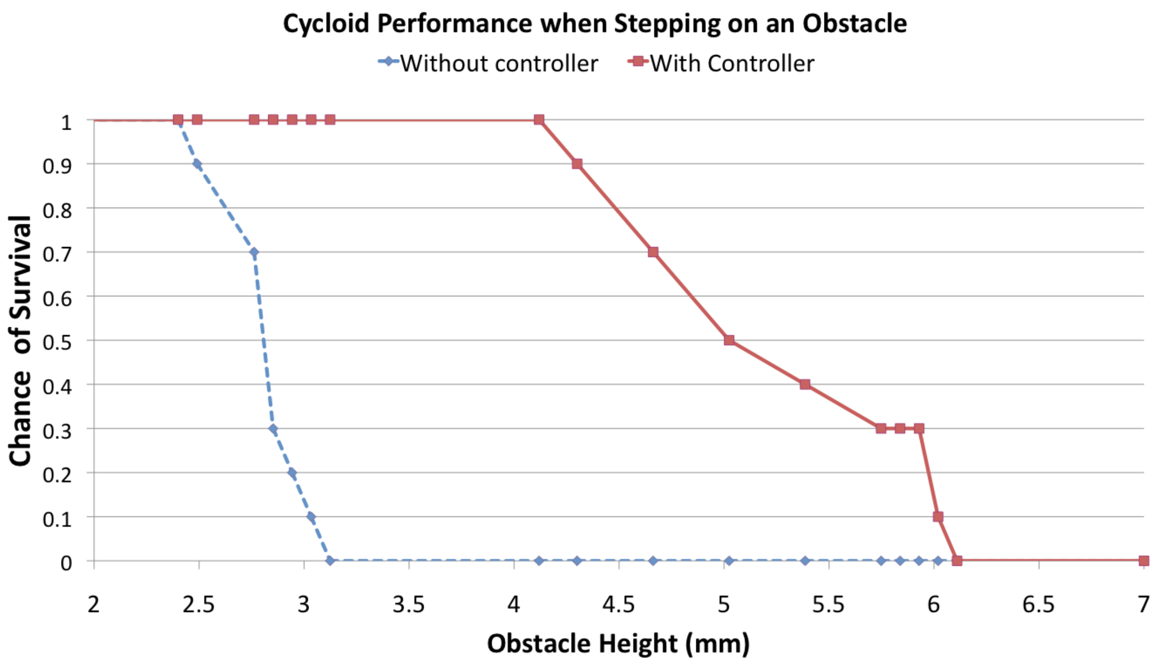


Figure 5.4.: Comparison of the manageable height when stepping on an obstacle with and without the pendulum controller enabled. [HLW11]

of a grass felt. This might be the case because the felt got the legs caught when the robot tried to take large steps whereas they could still continue their move on a plastic surface sliding on the ground. This has not been investigated further as it happened in such cases where the motors could not reach their supposed positions in time even when they were operating at their maximum-performance which then resulted in making the leg movement uncontrollable in any case. The red lights on the foot-sensor boards 5.7 were used to signalize whenever one or more motors were over 12° off from where they should be.

## 5.4. Collision test

Another experiment shows the ability of the system to adapt to a situation even when this forces it to do the opposite to what its goal wants it to do, as the higher priority is to keep the robot balanced upright. For this experiment the robot was made to walk forwards on a collision course against a chair 5.5. The chair was moved towards the robot by a human with about the same speed as the robot moved towards the chair. The chair is obviously the stronger opponent thus the robot would have to give in to keep balanced.

The diagram 5.6 shows the systems phase very small in blue and the pendulumX value in red. Whenever the phase is high, the left leg is the swing-leg and the right leg is the support-leg and vice versa when the phase is low. The diagram shows clearly distinct areas in which the average pendulumX is either more negative or more positive, similar to the diagram from the quick policy change. But in this scenario the policy always stayed the same, telling the robot to walk forwards.

While the robot quickly fell over when the pendulum controller wasn't used, it kept stable and walked backwards adapting to the chairs direction when using the pendulum controller. As soon as the chair was moved away again it immediately walked forwards again. This shoving

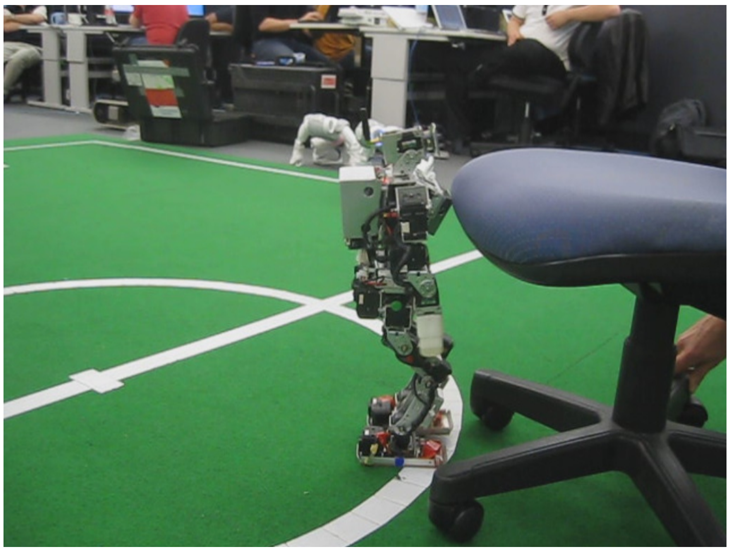


Figure 5.5.: The robot colliding with a chair. The robot being the weaker opponent has to give in and follow the chairs direction.

with the chair and then letting the robot walk again was repeated multiple times which can be seen in the diagram. The regions with the pendulumX in average being more negative indicate that the robot got shoved backwards whereas the more positive areas show when it was freely walking forwards again. This shows that by the use of the pendulum controller the robot adapts flexibly to the situation instead of falling over. Of course it is only able to adapt to velocities which are not much faster than the speed at which it can walk on its own.

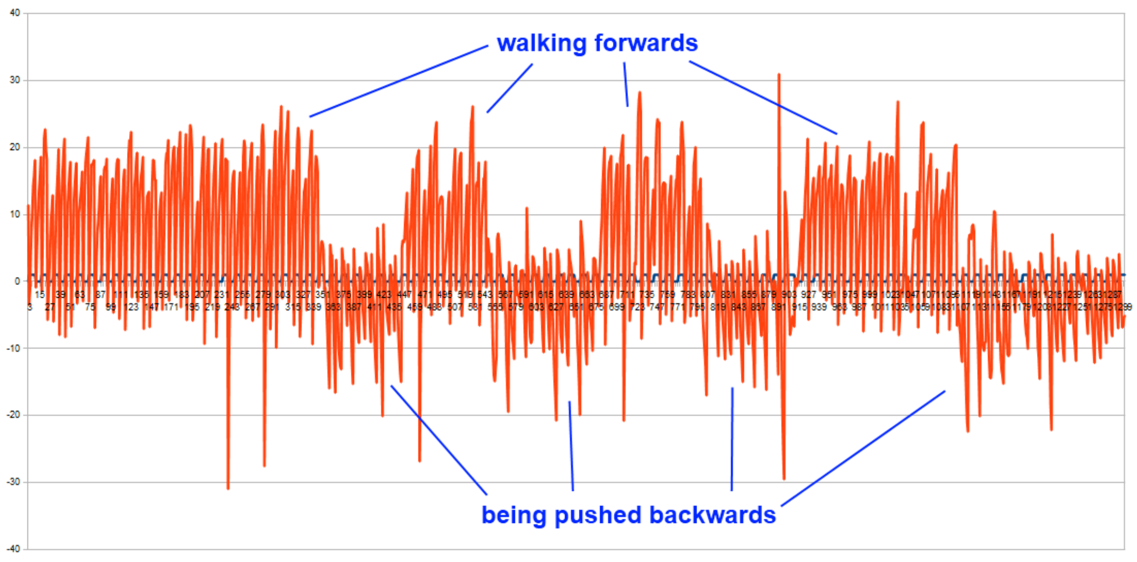


Figure 5.6.: The different regions show when the robot walks forwards and when it is following the chairs direction walking backwards.

## 5.5. Chapter summary

Overall, the tests showed that using the pendulum controller made the robot more stable and flexible. A routine that checks that the actual motor positions comply with the set positions lights up the foot-sensor boards LEDs 5.7 whenever any motor is out of compliance by more than 12°. This has been implemented to directly see whenever the motors are struggling during the experiments. In the last two of the three experiments the LEDs lit up multiple times showing that the robot was driven to the limits of its hardware.

The Quick Policy Change test showed that the system is able to switch its goal spontaneously no matter in which part of the walk-cycle it currently is. Other systems might have to send commands to decelerate, stop and accelerate again to realize this. This system just changes the goal variable which leads to another region of the RL learned controller being read from the controller array. The change in velocity happens on its own without further calculations.

The results of the stepping tests were already good. To improve the results further, some technique that either stops the step from being taken, or adapts the placement to the ground could be developed. Additionally, the lights on the foot-sensor boards lit up many times at the higher steps indicating that the motors were struggling to follow. They always lit up before the robot fell when it was about to fall.

The same happened at the collision tests when the robot moved forwards and was shoved backwards at a speed which was on the limit of the actuator's abilities. This leads to the assumption that the pendulum controller system would be capable of more if it was used on a robot with stronger and faster actuators.

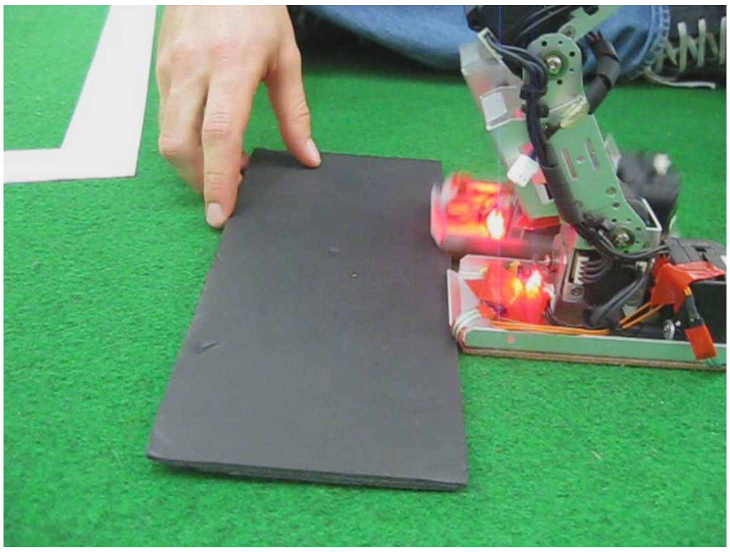


Figure 5.7.: The LED lights on the foot-sensor boards are on in that moment, indicating that one or more motors are off by more than 12° from where they should be.

# 6. Summary and future work

## 6.1. Summary and future work

The Cycloid II robot has successfully been set up as a research platform by programming different modules to control its actuators and observe its sensors. The platform was then used to try out and evaluate a technique to balance biped robots. The inverted pendulum model used to represent the biped's balance and actuate ankle control to influence the balance was implemented on the system. The tests which were conducted worked out very well 5. In the beginning the system estimated its state only using the observed motor positions and the foot-sensor values, neglecting the IMU values, which already worked very good. This shows that the state estimation already works well enough without the IMU to get reliable results.

Figure 6.1.: Darwin, successor of the Cycloid II [rob11].

The developed pendulum controller system showed promising results and would therefore be worth to be improved in multiple directions in the future. One of the next steps could be to extend it to omnidirectional motion by including coronal movement. Thereby the system would not only be extended in its scope but also be improved in its overall stability as the side-to-side rocking motion can lead to a fall with a sideways component if the robot is disturbed while it is on one foot only, which is the case in most of the time during motion. If omnidirectional movement would be integrated, the state space would have to grow but there are ways to handle this, "[...] such as more cost-effective function approximation using instance based methods [SSR97] and hierarchical techniques [RB03]." [HLW11]. The double support phase where both feet are on the ground is not being handled as such in the current implementation, which assumes that there is always one support-leg and one swing-leg. But as there is an increase in stability while the robot has both feet on the ground, ways could be developed to make use of this.

Although the simulation suggested that a direct control of the swing-leg would not improve the balance much further when the foot itself is big enough, conducting more research in that direction might make it possible to reduce the size of the over proportionally big feet without the loss of stability.

Later on it would be interesting to distinguish more between the state estimated via the motors and foot-sensors and the state estimated via the IMU which could make it possible to walk from an even ground onto an incline surface without triggering the balancing but by adjusting the angle of the foot instead.

It would be interesting to test the performance of the pendulum controller on stronger hardware like the Darwin robot from robotis[rob11] which is the successor of the Cycloid II. The Darwin is about the same size as the Cycloid II and uses a 1.6 GHz Intel Atom processor with a 4GB on-board flash SSD but the key difference would be the newer motors which are able to perform a faster walk, which might improve balancing skills.

## 6.2. Related work [1]

Much work has already been done to make biped robots walk and different versatile approaches were chosen. Reaching from an approach using function approximation which is similar to neural networks [BF97] to undertakings with an actuated passive walker controlled by frontal plane control [TZS04] and other approaches also using RL techniques to control the foot placement [MCAZ04] [WBBH06] [MNE$^{+}$05]. The Humanoid Robotics Project [Wikb] is also tracking the CoM and additionally the Zero Moment Point (ZMP) that they use in a modern control theory which they developed using preview control that is planning ahead [KKK$^{+}$03] [KMM$^{+}$10]. Analytic methods which also use an inverted pendulum model, make use of iterative calculations to control the swing leg placement [GR10]. In "Reinforcement learning for a cpg-driven biped robot" [MNaSI04] a policy gradient method is used to learn the parameters of a central pattern generator and in a paper of Chee-Meng Chew et al. [CP02] the parameters of a swing leg policy are learned using a CMAC function approximation.

[1]As it is related to the same project, the related work section of our paper [HLW11] is similar to this one.

# 7. Appendix

## 7.1. Continuing research on the Cycloid II

This chapter is especially useful for anyone continuing to research with the Cycloid II. It is explained how to handle the problems which sometimes occur during start up and how the system is quickly set up to be used. Furthermore, known issues and how to cope with them is explained, followed by a list of useful software and links to it.

### 7.1.1. Starting the robot

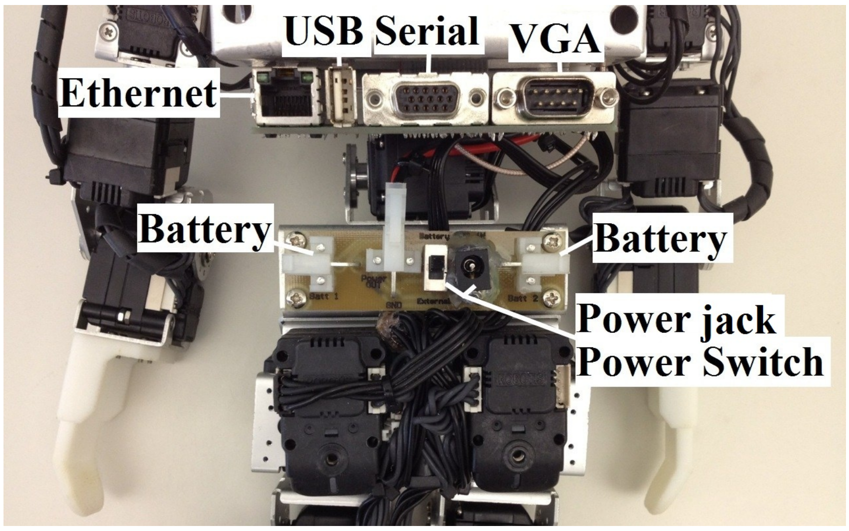


Figure 7.1.: The back of the robot shows the ports for the Ethernet, Serial, VGA and USB interfaces. Below them are the two battery connectors, the external power connector and the switch to change between external power and battery.

- Connecting the batteries and/or AC and selecting the power source on the switch will start the system. (If both sources are plugged in, then the source can be changed without a restart as there is no off position on the switch).
- If Linux doesn't boot up automatically and the robot stops after the Bios screen, the following will probably help:
  - Connecting a serial cable and opening a HyperTerminal like putty with the standard settings (8 Data bits, 1 Stop bit, no parity and XON/XOFF as Flow control) at a baud rate of 38400. Starting the robot now will show the start up screen in the terminal and make it possible to go into the bios via the terminal. In the bios, set the boot device to: Usb Harddrive and under → Features → Console Redirection to

auto.
It should now start from the Usb Stick at the internal Usb port.

### 7.1.2. Linux system

The desktop environment Xfce is installed and can be started using the command: ”startx”.

The robot was last used in the robot lab at CSE on Level 3 and it is set up to connect to the robocup WirelessLAN automatically using the interface wlan2, if it doesn’t log in automatically restarting the wireless might help:

”ifdown wlan2” followed by ”ifup wlan2” and check that the network is still the same.

The best way to control the machine is the ssh server which is automatically started. To compile code on it, no special packets are needed but for the code we used, the libftd2xx.so.1.0.4 driver has been used to communicate with the motors, foot-sensors and the HySense lite module which are all connected to different Serial (to USB) ports.

The driver can be found at: `http://www.ftdichip.com/Drivers/D2XX.htm`

Installation instructions at: `http://wikiri.upc.es/index.php/Communications_library`

The driver showed problems which randomly stuck the whole system for a second when all three serial port devices were used in 10ms cycles. This only happened when the code using the *libftd2xx* driver was started as root and thereby able to change the process priorities. In that case the driver starts some threads with highest priorities which freeze the system for a second every now and then. The solution is to start the code as a normal user, for example the student login. To be allowed access to the Serial ports as a normal user, the following command has to be executed as root:

”chmod o+w /dev/bus/usb/001/*” when the system boots up it occasionally assigns the serial ports differently under ”/dev/bus/usb/002”, then

”chmod o+w /dev/bus/usb/002/*” has to be executed instead.

Code can also be compiled on another Linux machine and be transferred afterwards using rsync. An easy rsync set-up script can be found here: `http://kevin.vanzonneveld.net/techblog/article/synchronize_files_with_rsync/`

### 7.1.3. Using the walking code

The latest compiled code can be found on the robot in: /home/student/robotCommands/. ”noControllerRock” & ”noControllerSlowlyBackwards” & ”noControllerSlowlyForwards” are the files which let the robot do the actions they are named after without the use of any controller. ”controllerBackAndFor” & ”controllerJustRock” use the ”controller 06.06.txt” to do the actions.

The source code is mostly self-explanatory. Very important are the Motor null Positions which are found in the top array in the Motors.cpp Class. Slight changes can make the robot unstable/more stable and make it walking forwards/backwards slowly when it should just rock. It seems that the joints can change a small bit just from one day to another whereby the null positions of the ankle may have to be adjusted. The picture of the robot with its motors and IDs on them shows which number each motor got assigned 2.4(a). Developing a filter which adjusts this might help to solve that issue. The source code is found on a CD added to this instructions.

Commented lines ("//") which were for testing or remembering purposes only can still be found in the code but do not need to stay there. The "main loop" 4.4 can be found in "Controllthread.cpp", there are also a few commented "cout" lines which were just for testing purposes. Some of these variables are made global just for the purpose of quickly getting them into the logging and to print them to the console.

### 7.1.4. Known issues

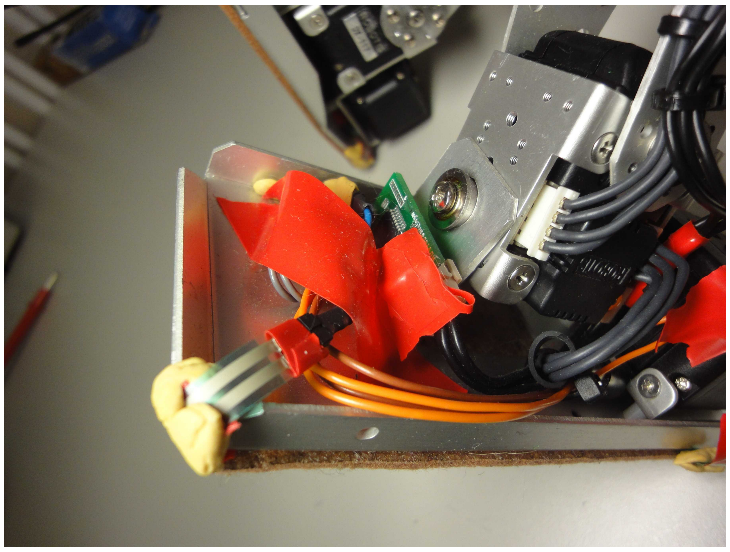


Figure 7.2.: Foot-sensor board attached to the foot, protected by hot glue and tape over it that it doesn't short circuit when the ankle tilts extremely upwards.

- The foot-sensor boards, shown in figure 7.2 can touch the robots lower legs which would result in a short circuit when tilting the feet totally upwards. To prevent this, some hot glue covered by tape was put on the sensor boards to protect them. This hot glue might possibly liquefy if the LEDs on the board would be lit up continuously for a while. If they shall be lit up continuously, then this should be monitored or the boards could be protected by putting some tape on the legs instead. Still, this issue would not come up in normal use, just if something goes wrong while conducting tests.

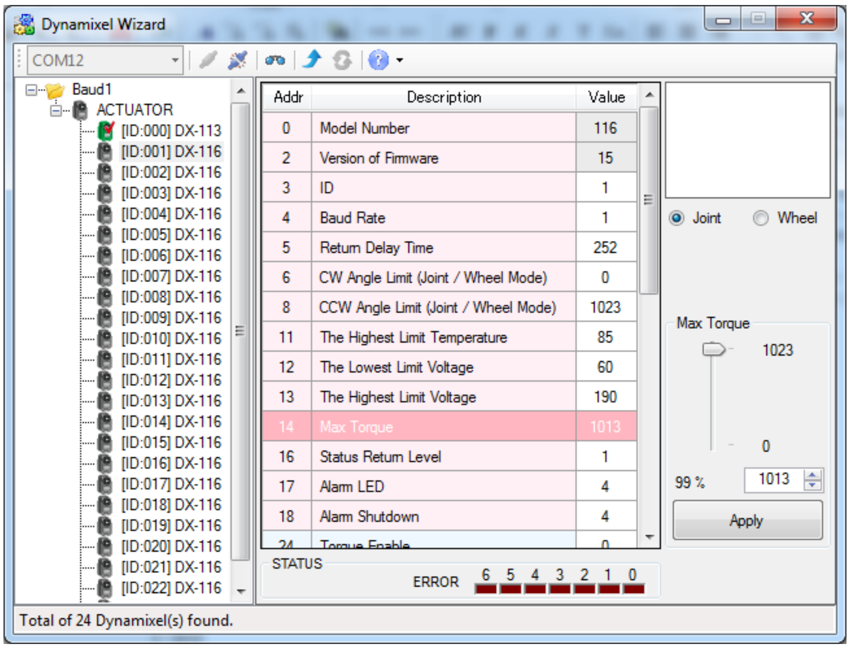
- If the robot falls over, it is unfortunately likely to reboot. This is probably caused by the uppermost mainboard in the back of the robot which is the power supply board. As it is only fixed by one screw in one corner, the first thing to do if something doesn't work as intended is to press it down with the back of a screwdriver through one of the air holes in the back of the cycloid.

- When the robot reboots after falling over it sometimes can't communicate with the HySense module anymore. Rebooting it a couple of times or keeping it turned of for a while helps. It also happens that the time resets, causing Linux to check the filesystem and reboot.

- The force sensor pad's connections are breaking sometimes. Therefore it is recommended to protect them with tape and plasticine similar to the protection on the existing ones on the robot. Replacement sensor pads can be found in one of the boxes for the Cycloid II. Although that they are bigger they work just as good.

- Screws are getting loose every now and then. I've unscrewed and put thread locker to all the screws in the legs and to some in the torso and in the arms. It still happens that for example the screws in the hip motors get loose as they have to handle heavy usage while often holding the whole robot. If they continue to loosen, buying a stronger thread locker should be considered.

### 7.1.5. Software used



| Addr | Description | Value |
|---|---|---|
| 0 | Model Number | 116 |
| 2 | Version of Firmware | 15 |
| 3 | ID | 1 |
| 4 | Baud Rate | 1 |
| 5 | Return Delay Time | 252 |
| 6 | CW Angle Limit (Joint / Wheel Mode) | 0 |
| 8 | CCW Angle Limit (Joint / Wheel Mode) | 1023 |
| 11 | The Highest Limit Temperature | 85 |
| 12 | The Lowest Limit Voltage | 60 |
| 13 | The Highest Limit Voltage | 190 |
| 14 | Max Torque | 1013 |
| 16 | Status Return Level | 1 |
| 17 | Alarm LED | 4 |
| 18 | Alarm Shutdown | 4 |
| 24 | Torque Enable | 0 |

Total of 24 Dynamixel(s) found.

Figure 7.3.: Screenshot of the Dynamixel Wizard which is very useful for initial configurations of the motors e.g. setting the IDs and trying out a motor's behaviour for certain values.

- The RoboPlus software designed for the Dynamixel Wizard to configure the motors: `http://support.robotis.com/en/software/roboplus_main.htm` (RoboPlusWeb(v1.0.20.0).exe can be found on the CD I burned and left.)
- Manual for the USB2Dynamixel adapter: `http://www.tribotix.info/Downloads/Robotis/USB2Dynamixel/USB2Dynamixel_manual%28english%29.pdf`
- Cycloid II at robotics: `http://www.tribotix.com/Products/Robotis/Humanoids/CycloidII_info1.htm`
- FTDI drivers page: `http://www.ftdichip.com/Drivers/D2XX.htm`
- Actuator DX-117 specifications: `http://support.robotis.com/en/product/dynamixel/dx_series/dx-117.htm`
- Robotis Dynamixel homepage: `http://www.robotis.com/xe/dynamixel_en`

- Robotis Dynamixel Software: `http://support.robotis.com/en/software/dynamixel_sdk/usb2dynamixel/usb2dxl_linux.htm`
- Tribotix Dynamixel software: `http://www.tribotix.info/Downloads/Robotis/Robotis.htm`
- Tribotix Wiki about HyInt: `http://www.tribotix.info/wiki/index.php/HyInt_v2_User_Manual`
- Foot-sensors: `http://www.bioloid.info/tiki/tiki-index.php?page=Foot+Pressure+Sensor+Manual` `http://www.huvrobotics.com/shop/index.php?_a=viewProd&productId=4`
- IMU Sensor-Board: `http://www.bioloid.info/tiki/tiki-index.php?page=6-Axis+Bus+IMU+Documentation` `http://www.sparkfun.com/products/9058` `http://www.sparkfun.com/products/9268`

# A. Glossary

| | |
|---|---|
| ZMP | zero-moment-point |
| CoM | center-of-mass |
| CoP | center-of-pressure |
| HRP | Humanoid Robotics Project |
| RS485 | An interface standard for digital communication. |
| SPL | The RoboCup Standard Platform League [Roba] |
| MCU | Microcontroller Unit |
| TTL | Transistor-Transistor-Logic, described here is an Interface with TTL signals operating in the range of 0V to 5V. |

# References

[BF97] Hamid Benbrahim and Judy A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283 – 302, December 1997.

[Big] Boston dynamics, bigdog - the most advanced rough-terrain robot on earth, 2011. `http://www.bostondynamics.com/robot_bigdog.html`.

[Com] CompuLab. Developer resources for cm-iglx, 2011. `http://www.compulab.co.il/iglx/html/iglx-developer.py`.

[Com09] CompuLab. *CM-iGLX Computer-On-Module Reference Guide*, June 2009.

[CP02] Chee-Meng Chew and Gill A. Pratt. Dynamic bipedal walking assisted by learning. *Robotica*, 20(5):477 – 491, September 2002.

[foo] Huv robotics, foot pressure sensor board, 2011. `http://www.huvrobotics.com/shop/index.php?_a=viewProd&productId=4`.

[GR10] Colin Graf and Thomas Röfer. A closed-loop 3d-lipm gait for the robocup standard platform league humanoid. In C. Zhou, E. Pagello, S. Behnke, E. Menegatti, T. Röfer, and P. Stone, editors, *Proceedings of the Fifth Workshop on Humanoid Soccer Robots in conjunction with the 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, USA, December 2010.

[HLW11] Bernhard Hengst, Manuel Lange, and Brock White. Learning ankle-tilt and foot-placement control for flat-footed bipedal balancing and walking. 11th IEEE-RAS International Conference on Humanoid Robots, Bled, Slovenia, October 2011.

[KKK$^+$03] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620 – 1626, Taipei, Taiwan, September 2003. Ieee.

[KMM$^+$10] Shuuji Kajita, Mitsuharu Morisawa, Kanako Miura, Shin'ichiro Nakaoka, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, and Kazuhito Yokoi. Biped walking stabilization based on linear inverted pendulum tracking. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4489 – 4496, Taipei, Taiwan, October 2010. IEEE.

[MCAZ04] Jun Morimoto, Gordon Cheng, Christopher G. Atkeson, and Garth Zeglin. A simple reinforcement learning algorithm for biped walking. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 3030 – 3035, New Orleans, LA, USA, May 2004.

[MNaSI04] Takeshi Mori, Yutaka Nakamura, Masa aki Sato, and Shin Ishii. Reinforcement learning for a cpg-driven biped robot. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 623 – 630, San Jose, California, 2004. AAAI Press.

[MNE+05] Jun Morimoto, Jun Nakanishi, Gen Endo, G. Cheng, C.G. Atkeson, and G. Zeglin. Poincaré-map-based reinforcement learning for biped walking. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2381 – 2386. IEEE, April 2005.

[RB03] Balaraman Ravindran and Andrew G. Barto. Smdp homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 03)*, volume 18, pages 1011–1018. Citeseer, 2003.

[Roba] Robocup. Board of trustees, 2011. `http://www.robocup.org/`.

[Robb] Aldebaran Robotics. Nao, 2011. `http://www.aldebaran-robotics.com/`.

[Rob05] Robotis. *Dynamixel DX-113, DX-116, DX-117 Users Manual*, 2 edition, November 2005.

[rob11] Robotis, robot company. `http://www.robotis.com/xe/`, 2011.

[SB98] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *MITPress, Cambridge, Massachusetts*, 1998.

[sci] Metralabs mobile robots, scitos a5, a versatile mobile service-robot, 2011. `http://metralabs.com/index.php?option=com_content&view=article&id=67&Itemid=66`.

[spf] Sparkfun electronics imu analog combo board, 2011. `http://www.sparkfun.com/products/9268`.

[SSR97] Juan C. Santamaria, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163 – 217, September 1997.

[SVV64] L. M. Sonneborn and F. S. Van Vleck. The bang-bang principle for linear control systems. *SIAM J. on Control and Optimization*, 2:151 – 159, 1964.

[Tik] TikiWiki. Foot pressure sensor board manual, 2011. `http://www.bioloid.info/tiki/tiki-index.php?page=Foot+Pressure+Sensor+Manual`.

[Tri11] Tribotix. an australian robot company selling and upgrading robots from the robotis company. `http://www.tribotix.info/`, 2011.

[TZS04] Russ Tedrake, Teresa Weirui Zhang, and H. Sebastian Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2849 – 2854. IEEE, 2004.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 7(1), 1995.

[WBBH06] S. Wang, J. Braaksma, R. Babuska, and D. Hobbelen. Reinforcement learning control for biped robot walking on uneven surfaces. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 4173 – 4178, Vancouver, BC, Canada, July 2006.

[WGC$^+$07] Eric R. Westervelt, Jessy W. Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*, volume 1. CRC Press, Boca Raton, 2007.

[Wika] Wikipedia. Atan2, 2011. `http://en.wikipedia.org/wiki/Atan2`.

[Wikb] Wikipedia. Humanoid robotics project, 2011. `http://en.wikipedia.org/wiki/Humanoid_Robotics_Project`.