Bipedal walking

Martin Wicke

May 2001

Author:	Martin Wicke
Supervisor:	Thomas Bräunl, Michael Kasper
Organisation:	Universität Kaiserslautern
	Fachbereich Informatik
	AG Robotik und Prozeßrechentechnik
	Postfach 3049
	67653 Kaiserslautern

Contents

List of Figures

Chapter 1

Introduction

This thesis presents an attempt to implement walking on a biped robot, "Johnny Walker" (see figure 3.1 (b) on page 16). The robot was constructed in 1998 with the goal to provide a low-cost autonomous biped robot ([1]).

Bipedal locomotion is of great interest in the field of robotics. The idea of building a humanoid robot is appealing and the ability to walk is a vital requirement. Certainly, the interest in legged and especially two-legged locomotion is old, but only recently the necessary technology to build and control a complex process like this has become feasible.

On a first glance, two-legged walking does not seem very useful for robots. In nature, it allows humans to have their hands available at all times, they can reach higher and have a better view over their environment. In robotics, these characteristics are no real benefits: if a robot needs to reach higher, one would mount longer arms, if it needs manipulators available, it can be equipped with another one. There are no hardware requirements that make bipedal locomotion for robots necessary.

An argument that is often used to justify the need for two-legged walking is the high flexibility. Legged machines can handle certain types of obstacles much better than wheeled vehicles can. On bumpy terrain or on grid structures, multi-legged walking machines have proven to be very effective. But still, no need for the far more complex two-legged walking can be seen.

The motivation to study two-legged locomotion is directly based on ourselves being two-legged walkers. Humans have customized their environment to suit their needs as bipedal walkers, e.g. by inventing constructions like stairs, which can be unsurmountable obstacles for other types of vehicles, especially wheeled ones. Of course, in an environment that is built to be convenient for humans, humanoid robots have the best qualifications.

Another argument is based on psychology. In the future, robots are supposed to be used for service tasks. Obviously, humanoid robots would be accepted far easier than other types of robots, particularly if their duty involves interaction with humans (just imagine an giant arachnid serving your tea).

Before a bipedal machine can be of any use, it has to walk. Since walking itself is very challenging and complex (complex enough to keep all resources busy), biped walking machines currently do not perform any other tasks.

1.1 State of the art

The presumably most advanced biped robot currently existing is the Honda P3 ([2, 3]). The Honda P3 is a humanoid autonomous robot that can walk, climb slopes and stairs. It has 28 degrees of freedom and stands 160 cm tall. The joints are driven by DC-motors and the robot can operate for 25 minutes with the batteries carried. To reduce weight, the aluminium skeleton used in its predecessor P2 has been replaced by magnesium. Figure 1.1 shows the robots.

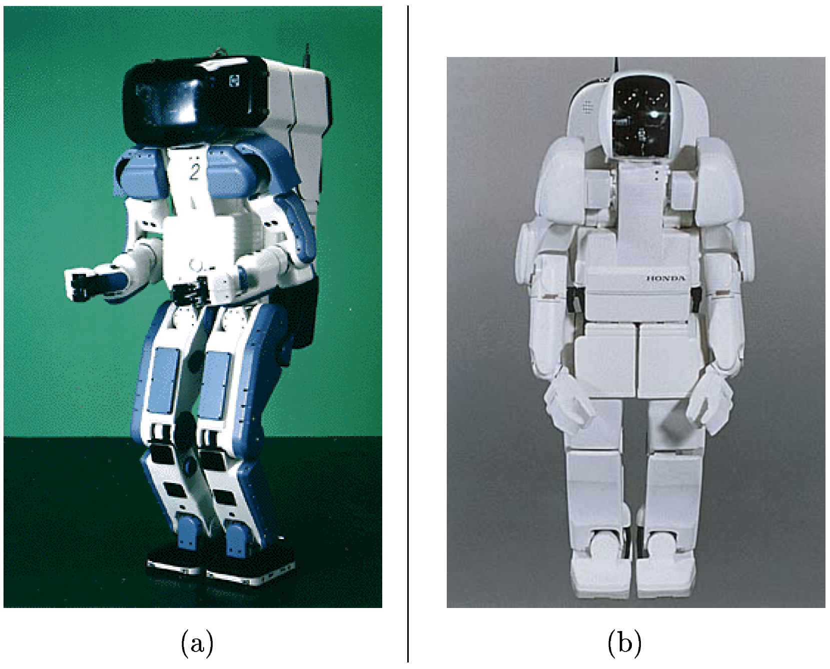


FIGURE 1.1: The Honda P2 (a) and P3 (b) humanoid robots

Another robot worth mentioning is the Shadow robot, built by the Shadow Robot Group. This robot uses air muscles as actuators, instead of the more commonly used DC motors and has a wooden chassis. It mimicks the physiology of human legs very closely, which is only possible using actuators of similar shape and size as human muscles. The Shadow robot is not able to walk. One of the research objectives is to gain knowledge that can be used in prosthetic design, hence the human-like construction. For further information see [4].

Of course, there are plenty of experimental bipedal walking machines, some of them humanoid. We will call the robot “Johnny Walker” used for the experiments herein humanoid, even though it has far less degrees of freedom than a human and is lacking arms. The robot will be presented in detail in chapter 3.

1.2 Thesis layout

After this introduction, chapter 2 will deal with some mathematical models that are used to descibe walking and help understanding the walking process. In publications on bipedal walking the different tools are frequently confused. Therefore, the discussion of walking theory is quite extensive. Chapter 3 presents an actual gait that was developed for “Johnny Walker”. Possibilities of improvement will be discussed and a different approach to gait generation will be presented that could help to overcome the problems faced. In chapter 4, I will introduce a camera-based inclinometer using an artificial horizon. It can be used as a replacement for or in cooperation with the acceleration sensors. Finally, chapter 5 will summarise the results and an outlook on what can and should be done in future will be given in chapter 6.

Chapter 2

Bipedal locomotion

Two kinds of biped locomotion can be distinguished: walking and running. In a walking gait, the robot will always have contact to the ground with one of its legs. When running, it will leave the ground completely, making a running gait a series of jumps. This thesis will only deal with walking. A running robot was presented by Chevallereau and Sardin in [5]. The robot does not run freely but is mounted on a guidance device which allows only movement in a circle. Therefore, only falling in one direction (forth or back) has to be considered. A software and hardware design for a biped running robot was done by Gienger et al. in [6]. However, the robot is not yet built.

Several mathematical tools help to understand the walking process: in this chapter, the center of gravity (CoG), the zero moment point (ZMP) and the foot rotation indicator (FRI) will be adressed. All these are tools that give information about the robot's stability. They can be exploited to plan a stable gait or to correct a gait online. Control strategies will be discussed in chapter 3. For the following considerations, a cartesian coordinate system as shown in figure 2.1 will be used.

A simple and reasonable assumption on the robots contruction can be made: The robot's body consists of a number of rigid segments connected by joints. This allows us

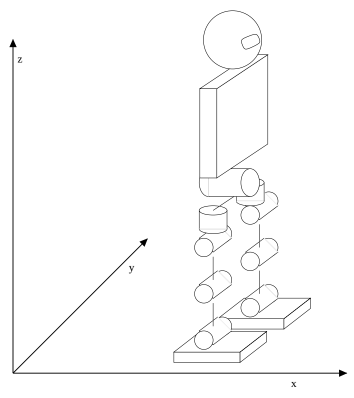


FIGURE 2.1: The coordinate system used in all further calculations

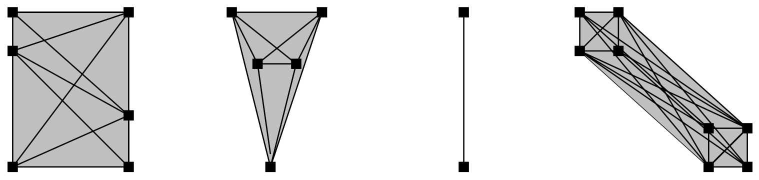


FIGURE 2.2: Sets of points and their convex hull: A straight line connecting any two points in a set always lies inside the convex hull

to fully describe the dynamics of the robot by knowing the position and movement of the segments' centres of mass (CoM). The only friction forces that will be considered are the ground friction forces. We will assume that these forces are always great enough to prevent sliding (a dangerous assumption, as we will see).

2.1 Centre of pressure and area of support

When analysing the robot's stability, the term *supported polygon* or *area of support* will appear. Both synonyms describe the convex hull of all the robot's ground contact points (see figure 2.2). For any set of points S, the convex hull of S is the set given by:

$$\left\{ \sum_{P \in S} (\lambda_P \cdot P), \text{where } \lambda_P > 0 \; \forall P \wedge \sum_{P \in S} \lambda_P = 1 \right\} \tag{2.1}$$

The forces acting on the robot are intertia, gravity and ground reaction forces. Gravity and intertia forces can make the robot fall, the only force able to prevent this is the ground reaction force. Most of the following deals with the question under which circumstances the robot is balanced, i.e. there is an equilibrium between ground reaction forces on the one hand and gravitational and inertia forces on the other hand.

Therefore it is crucial to know where the ground reaction forces act. This point is called the *centre of pressure* (CoP) known from fluid dynamics. The CoP is the point on a plane surface submerged in a fluid where the upward pointing forces act. It can be calculated by integrating over the forces acting on the surface:

$$P = \frac{\int p_A \cdot f_A dA}{\int f_A dA} \tag{2.2}$$

Where in our case P is the position of the CoP, A is an area segment in the foot plane, p_A is the position of that area and f_A is the vertical ground reaction force acting on it.

For now, the only porperty of this point that we need to know about is the fact that it will always lie inside the supported polygon. Setting

$$\lambda_{p_B} = \frac{f_B}{\int f_A dA}$$

for all possible areas B, this becomes obvious. The CoP is interesting to us in many ways and will be discussed further with the ZMP in section 2.3.

2.2 Centre of gravity

The *centre of gravity* is the projection of the robot's CoM in direction of gravity onto the ground plane. We determine the overall CoM by calculating the weighted sum of all segments' CoMs:

$$C = \frac{1}{M} \sum_i c_i \cdot m_i \tag{2.3}$$

Where C denotes the robot's CoM position, M is the robot's total mass, m_i is the mass and c_i is the CoM position of the i^{th} segment.

Since in our coordinate space, gravity always acts along the z-axis, the projection is quite simple and the CoG is given by

$$G = \begin{pmatrix} C_x \\ C_y \\ 0 \end{pmatrix} \tag{2.4}$$

Since only the position of the robot mass influences the CoG, it can only be a tool to determine static stability. This static analysis assumes that the robot is not moving, making gravity and ground reaction the only forces acting. Both forces act along the Z-axis, in opposite directions.

The CoG is the point on the ground plane where the ground reaction forces would have to act to neutralise the gravitational forces.

Whenever the CoG lies within the supported polygon of the robot, the robot is called statically stable. In this case, the ground reaction force can and will act in the same point (the CoP and CoG will fall together), completely neutralising the gravitational force. If the CoG is outside the supported polygon, the gravitational and ground reaction forces cannot act in the same line and a moment will occur that will eventually make the robot tip over. The distance of the CoG to the area of support is an indicator for the amount of static instability, since the occuring moment is proportional to the distance CoG-CoP.

Using the CoG, we can now distingiush between two kinds of gaits: static and dynamic walking. We call a gait static if the CoG lies in the area of support at all times, dynamic otherwise. Static walking gaits are commonly used by robots with four or more legs, since the area of support can always be quite large. Machines with less legs will have a dynamic gait, except when walking very slowly. Even a tripod cannot keep more than two feet on the ground all the time. A bipedal walker's area of support will be restricted to one foot during single support phase. When walking faster, the robot's inertia will become more important than the gravitational forces and the robot can easily fall even if its CoG is inside the area of support. Figure 2.3 shows static and dynamic walking gaits for different

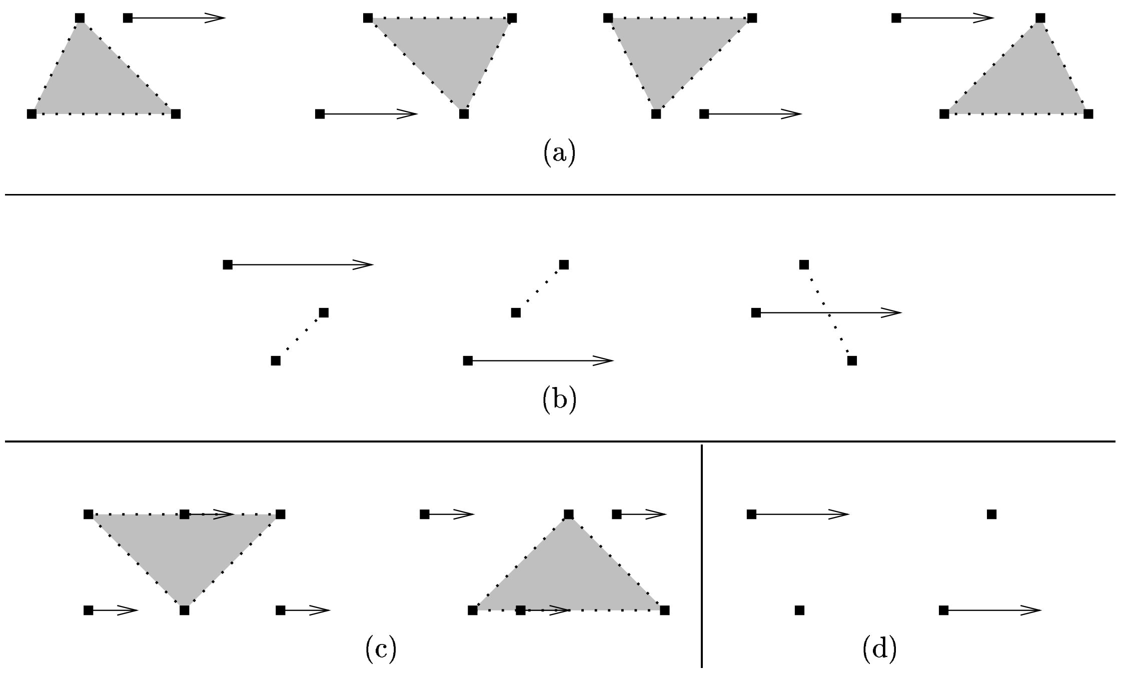


FIGURE 2.3: Walking gaits for quadroped (a), tripod (b), hexapod (c) and biped walking (d). Gaits a and c are static, b and d are dynamic gaits.

robot designs.

2.3 Zero moment point

The term *zero moment point* was first introduced by M. Vukobratovic in [7]. In literature about biped locomotion, the ZMP is probably the most commonly used tool. Even though the concept is now known for more than 30 years, it is frequently used in a wrong context. It was defined by Vukobratovic as the point where the resultant of the vertical ground reaction force penetrates the ground surface. This is also the definition of the CoP, which we discussed earlier. In fact, these two points are identical (as shown in [8]). Many papers claim that the ZMP has to be within the area of support during the gait in order to achieve walking (e.g. [9–11]). We know that this is a trivial postulation and these papers really refer to what we call FRI. This will be discussed in section 2.4. In this work, the ZMP will be used according to its original definition to prevent further confusion.

As could be seen in section 2.2, the CoG helps us understand static walking but becomes less and less useful as inertia forces increase. The ZMP is a different approach to stability. Contrary to the CoG, a ZMP analysis includes the dynamics of the robot, making it particularly useful for bipedal locomotion.

We already know that the ZMP is defined as the point on the ground plane where the vertical ground reaction force acts. This force could be relocated to any point on the ground, producing an angular momentum (as shown in figure 2.4). The ZMP is the only

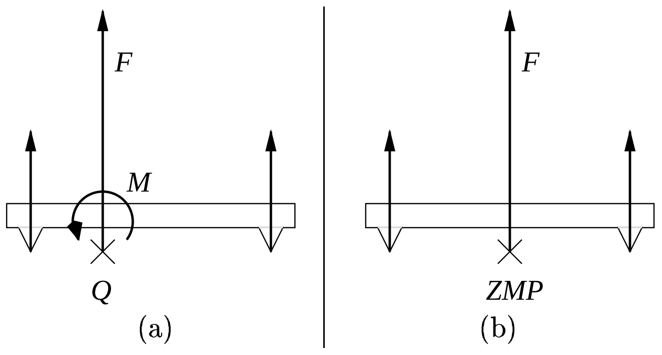


FIGURE 2.4:
(a) The Forces acting on the foot can be reduced to an equivalent force/moment pair acting in an arbitrary point Q.
(b) The zero moment point is the point where the resultant of the vertical ground reaction forces acts without a moment.

point where the momentum along both x- and y-axis are zero — hence the name zero moment point. There still can be a moment around the z-axis caused by frictional forces. Also, the resultant force acting in the ZMP does not have to be vertical due to friction forces.

Only little information can be extracted from the ZMP position. It will be inside the supported polygon, wether the robot is stable or not. If the robot is falling, the ZMP will be on the boundary of the area of support. The robot will then fall along the line through the ZMP and the centre of the supported polygon. Note that the robot is not necessarily unstable if the ZMP is on the boundary of the area of support.

The main use of the ZMP is to keep it moving along a predefined trajectory. An interesting approach was followed by Huang et al. in [12]. To achieve stability the ZMP was kept at a minimum distance from the boundary of the area of support. This way, it is possible to implement control for the ZMP (trying to keep it near the centre of the supported polygon).

2.4 Foot rotation indicator point

More interesting control can be done using the *foot rotation indicator* or FRI. This point was unknowingly used long before it was introduced and properly defined by A. Goswami in [8]. It is the dynamic analogy to the CoG. Recall the CoG being the point on the ground plane where the ground reaction forces would have to act to neutralise the gravitational forces. The FRI also considers inertia forces — it is the point on the ground plane where the ground reaction force would have to act to neutralise both gravitational and intertia forces. These forces are the only forces acting on the robot. Hence if the ground reaction force acts in the FRI, the robot is dynamically stable.

To find this point, we will compute the dynamic rotational equilibrium of the robot, calculating the moments around an arbitrary point Q:

$$M + (P - Q) \times R + \sum_i (G_i - Q) \times m_i g = \sum_i \dot{H}_{G_i} + \sum_i (G_i - Q) \times m_i a_i \qquad (2.5)$$

The left hand side of (2.5) represents the external forces gravity and ground reaction, the right hand side gives the internal moments. M denotes the frictional ground reaction moment, P is the position of the CoP, R is the resultant of the ground reaction force (acting in the CoP), G_i is the CoM position and m_i is the mass of the i^{th} robot segment. H_{G_i} is the angular momentum of the i^{th} segment around G_i and a_i is the linear acceleration of G_i. Equation (2.5) may be rewritten as follows:

$$\begin{aligned} M + (P - Q) \times R &= \sum_i \dot{H}_{G_i} + \sum_i (G_i - Q) \times m_i (a_i - g) \\ &\text{and} \\ (P - Q) \times R \sum_i Q \times m_i (a_i - g) &= \sum_i \dot{H}_{G_i} + \sum_i G_i \times m_i (a_i - g) - M \qquad (2.6) \end{aligned}$$

We are searching for one special point F in which the ground reaction force R should be acting to keep this equilibrium. This means that the moment caused by R around this point would be zero:

$$(P - F) \times R = 0 \qquad (2.7)$$

Using (2.7) in (2.6) we obtain

$$\sum_i F \times m_i (a_i - g) = \sum_i \dot{H}_{G_i} + \sum_i G_i \times m_i (a_i - g) - M \qquad (2.8)$$

We know that the gravitational acceleration g is directed downwards ($\Rightarrow g_x = 0 \wedge g_y = 0$) and the frictional momentum M acts only around the z-axis ($\Rightarrow M_x = 0 \wedge M_y = 0$). Using this in (2.8) leads to three equations:

$$\sum_i m_i \begin{pmatrix} F_y \cdot (a_{iz} - g_z) - F_z \cdot a_{iy} \\ F_z \cdot a_{ix} - F_x \cdot (a_{iz} - g_z) \\ F_x \cdot a_{iy} - F_y \cdot a_{ix} \end{pmatrix} =$$

$$\sum_i \begin{pmatrix} \dot{H}_{G_i x} \\ \dot{H}_{G_i y} \\ \dot{H}_{G_i z} \end{pmatrix} + \sum_i m_i \begin{pmatrix} G_y \cdot (a_{iz} - g_z) - G_z \cdot a_{iy} \\ G_z \cdot a_{ix} - G_x \cdot (a_{iz} - g_z) \\ G_x \cdot a_{iy} - G_y \cdot a_{ix} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ M_z \end{pmatrix}$$

Since we are searching for a point on the ground plane we can set $F_z = 0$ and write:

$$F_y = \frac{\sum_i m_i \left(G_y(a_{iz} - g) - G_z \cdot a_{iy}\right) + \sum_i \dot{H}_{Gix}}{\sum_i m_i(a_{iz} - g)} \tag{2.9}$$

$$F_x = \frac{\sum_i m_i \left(G_x(a_{iz} - g) - G_z \cdot a_{ix}\right) + \sum_i \dot{H}_{Giy}}{\sum_i m_i(a_{iz} - g)} \tag{2.10}$$

$$M_z = \sum_i m_i \left((G_x - F_x) \cdot a_{iy} - (G_y - F_y) \cdot a_{ix}\right) + \sum_i \dot{H}_{Giz} \tag{2.11}$$

Equation (2.11) describes the rotational equilibrium around the z-axis. The moments are balanced between the (frictional) ground reaction momentum and the internal moments. Since we assumed ground friction to be sufficent at all times, this is not interesting for us. If ground friction has to be considered, equation (2.11) determines the maximum momentum around the z-axis that the gait may produce.

The FRI position can be calculated using equations (2.10) and (2.9). If the FRI lies inside the supported polygon, it will be equal to the CoP. In this case, the ground reaction force actually acts where it should act and the robot will be stable. Just as the CoG, the FRI can leave the area of support, indicating an unbalanced moment. Its distance to the CoP is proportional to the unbalanced moment which can be computed using equation (2.12).

$$M = (P - F) \times R \tag{2.12}$$

Using this knowledge, it is possible to implement a control algorithm to keep the FRI inside the area of support.

A walking gait in which the FRI stays inside the area of support at all times will be called stable. Note that in a stable gait FRI and ZMP are always identical, probably one of the reasons for the confusion of these points.

2.5 Gait design

In order to walk, the robot must have a predefined gait it can follow. It is not yet possible to learn a bipedal walking gait from scratch — the most that can be learned so far are minor adjustments in the gait's parameters that allow the robot to adapt to changes in the environment. This section will adress the question how a basic gait can be synthesized.

Any walking gait should be stable. Even though non-stable walking is possible, it would be a constant process of losing and regaining balance. This condition introduces some contraints on the robot's movement. By constraining other properties of the gait one can narrow down the search for a suitable gait until one gait is chosen. Typical constraints include:

- Constant velocity of the robot (of the robot's CoM)

- Symmetry

- Soft foot impact
- Limited torques
- Smooth movements

All these restrictions will still not determine a gait. Thus, most gaits are designed to mimick the human walking motion. Following this thought, Dasgupta and Nakamura presented an attempt to generate motion patterns for a humanoid robot from human motion capture data ([13]).

Instead of reverse-engineering a gait from the constraints, one can as well start with a human-like gait and modify it step by step until it meets the requirements. This has the advantage that some of the neccessary modifications can be made online or learnt by the robot using its own sensory feedback.

A gait can be described as a set of functions, describing the position of each joint over time. No matter how the first basic gait is developed, it has to be made sure that it can be physically implemented on the robot. This means that position and/or torque control has to be available for each of the robots joints to make sure the joints follow their preset function. Of course, the joints have to be able to reach the angles the gait describes and the actuators must be able to supply sufficent torques.

A gait that is described by a set of parameters is easy to adapt to changing conditions. Values like step length or step height can be easily adjusted by learning algorithms. When these high-level semantics are not explicitly expressed in the formulation of the gait, this becomes virtually impossible.

A gait that was engineered using a model to meet certain requirements will of course not work as expected unless the model used to design the gait is completely correct. In most cases, a reliable and correct model is not available, since assumptions like elastic foot impact or the neglection of sliding cannot be held up. The next section will address these problems.

2.5.1 Gait generation with an inaccurate or lacking model

Often, the model used to calculate gait constraints does not reflect reality accurately enough. Any gait generated after these constraints will be faulty and may or may not work. If no better model is available, the gait has to be refined in a process of trial and error.

This can be done by reinforcement learning as shown by Chew and Pratt in [14]. Therefore, a Q-learning algorithm was allowed to change selected parameters determining the swing-legs motion. This intelligent control algorithm does not require any model of the environment. A simple reward function will give feedback after each step. If the hip velocity is within a desired range and the robot does not fall, the step is considered successful, a failure otherwise.

The greatest advantage of reinforcement learning is that it is unsupervised — no human "teacher" has to provide feedback. In this case, it is also model-free, meaning that no causal relationships between the algorithms choices (for gait parameters) and the effects on the environment (the robots actual behaviour) have to be known. These characteristics make reinforcement learning an ideal approach to control problems where only little is known about the behaviour caused by actions taken. Note that the robot is mainly controlled by conventional control mechanism, applying model-free intelligent control to all parts of the gait is not yet possible.

A different approach is to let a human "trainer" interact with the robot in order to stabilize its walking. In [15], a human-sized bipedal walking robot is outfitted with a handle. A human can apply forces while the robot is walking and thus stabilise the robot and prevent it from falling over. The applied forces are recorded. After a training cycle, a new gait is calculated, trying to mimimize the neccessary human intervention.

Setiawan et al. followed the same approach in [16]. The robot "WABIAN" used here is humanoid; instead of a handle the human teacher would guide the robot by its arms. In all publications, one has to be very careful what the exact interpretation of the term ZMP is.

2.6 Sensory feedback

It is obvious that stable walking can only be achieved if sensory feedback is available. Although one can synthesize a gait that will enable the robot to walk, the executed program can not react to any unforeseen events. Even basic control requires sensory information, but this kind of basic feedback is mostly included in the hardware and cannot be processed any further. It is reasonable to assume that without any feedback, bipedal walking would not be possible.

In order for a robot to be useful for any kind of task, it has to be capable of working under realistic conditions. This includes unpredictable events (e.g. a change of the ground condition) that have to be detected and compensated. Interesting features of the robot's environment that can be detected by sensor systems are the following:

- Posture feedback

  Under posture feedback we understand the possibility to detect the robot's joint positions. The easiest way to achieve this is using potentiometers. This kind of feedback allows us to control the movement of the robot, and to be confident that the actuators do reach the positions they are set to. This is a crucial requirement if a precalculated gait has to be closely matched.

  Using posture feedback it is also possible to calculate the CoG, if dimensions and mass of each of the robot's segments are known.

- Torso inclination

  Provided that the robot has a torso at all, torso inclination is a good indicator for falling. This data can be measured directly or it can be calculated from the leg's joint positions. To calculate it from the robot's posture one has to know the foot inclination. This angle can either be assumed zero if the foot rests flat on the ground or measured directly. One can easily verify the prior assumption e.g. by using mechanical or optical switches.

Measuring inclination angles means measuring the direction of gravity. This can be done by acceleration sensors, assuming that all accelerations except gravity are neglectable. This is a dangerous assumption and at least vibration has to be filtered using low-pass filters in either hardware or software. Even then, acceleration sensors turn out to be noisy and thus not very accurate (see [17]).

A different approach is to extract inclination angles from a camera image. This possibility will be discussed further in chapter 4.

- Orientation and position
  Odometry on a biped robot is far harder to implement than on a wheeled vehicle. It is useful to think about different approaches to calculating the robot's position and orentiation. Optical flow methods can be applied if a camera and enough computing power is available. Optical or other sensory data can be used to determine the robot's position using triangulation with artificial or natural landmarks. Position in a greater scale (down to some meters) can be obtained using a GPS-sensor. The easiest way to measure orientation is using a compass.

- CoP and FRI
  The centre of pressure can be calculated if the robot is outfitted with force or pressure sensors under the feet. This information can be used to implement a ZMP trajectory control. As soon as the CoP reaches the boundary of the supported polygon, no information about the degree of instability is contained in the CoP position.

  This can be solved by measuring the FRI. To do so, universal force sensors (UFS) are mounted in the lower legs, giving information about all moments and forces above this point. The only information missing is the foot data, which can be approximized or even neglected. This approach was carried out by Qinghua et al. and presented for example in [9]. Again, the term ZMP is used for what we call FRI.

  Using this kind of feedback it is possible to follow the gait and keep the robot stable even if it does not act exactly as the model used to design the gait predicts.

It can be seen that depending on the available sensory data, points like CoG, ZMP or FRI can be calculated online. Where the calculation of the CoG only requires posture feedback and some knowledge about the construction of the robot, measuring the ZMP requires foot force sensors; UFS are needed to calculate the FRI. Especially on a platform like the "Johnny Walker" robot, which is meant to be a low-cost experimental robot, it should be carefully evaluated which expenses can be justified.

Chapter 3

Generating a walking gait

For these experiments, the biped robot “Johnny Walker” was used. Former attempts to implement walking on this robot can be found in [1]. Using a gait that was designed to resemble the human walking motion, the robot was capable of taking up to four steps. Trials to increase the performance by adding feedback from acceleration sensors failed.

This chapter will present further effort on implementing stable walking using the experiences gained in the former experiments. Unfortunately, most of the programs written earlier could not be reused, since both hardware and operating system changed.

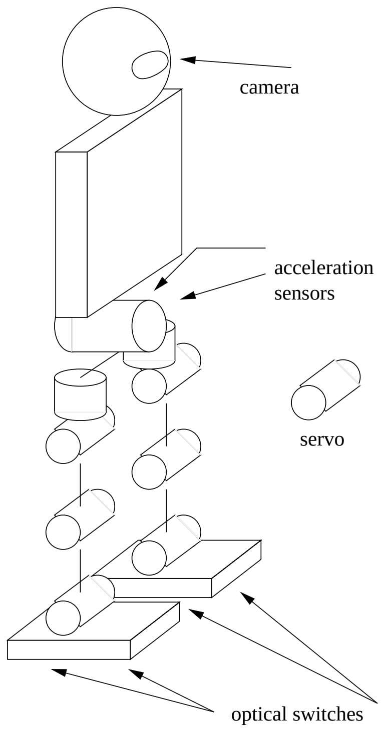
3.1 Johnny Walker

The “Johnny Walker” biped robot has nine degrees of freedom, driven by servos. Figure 3.1 (a) shows a schematic drawing. The robot is controlled by an Eyebot controller. Its sensors consist of a greyscale camera, two acceleration sensors and four optical switches. The relevant data can be found in [1].

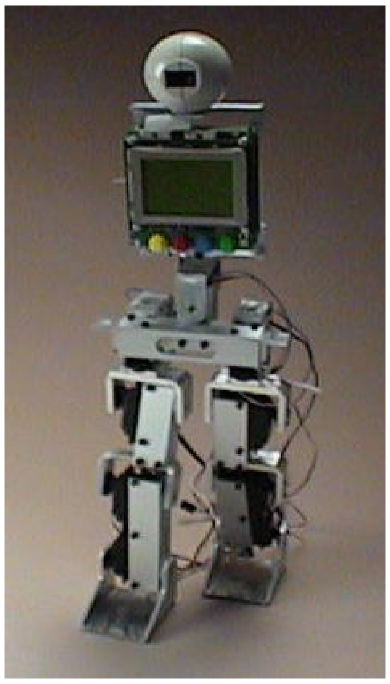
The optical switches were meant to detect foot-ground contact. They are specified to detect objects within 3 mm only, but they actually act as proximity sensors and switch whenever the ground comes nearer than approximately 2 cm. As the robot hardly ever lifts its feet above this height, these sensors are not used for feedback.

As explained earlier, acceleration sensors are quite noisy. In [17], Pepper dealt with improving the quality of the data obtained from the acceleration sensors. Despite these efforts, measuring the absolute torso tilt angle using the acceleration sensors is not workable. The sensors in use are two analog 1-axis accelerometers. As such, they are very sentitive to voltage fluctuations. The high-torque servos used in the robot produce high currents whenever it is moving. It turns out that none of the batteries used is capable of sustaining a stable voltage under these conditions, even a generic stationary power supply couldn’t. Thus, no valid accelerometer values can be read while the robot moves. This could be fixed by seperating the servo’s from the controller’s power supply.

The camera is the most versatile sensor available. The greatest problem is the high processing power that is needed to extract the information from the camera images. In



(a)



(b)

FIGURE 3.1:
(a) Drawing of the Johnny Walker biped robot. Shown are joints and sensor locations.
(b) A photo of the robot.

chapter 4, a possible replacement for the acceleration sensors using the camera is presented.

Note that no postural feedback is available. The servos have simple integrated feedback that allows them to control their position. The built-in controller decreases the applied torque when the actual servo position comes closer to the desired position. Thus the servo will in most cases never reach the position it is actually set to, since there are almost always forces acting against the servo torque. The remaining error between designated and actually reached position is determined by the force acting against the servo torque. Not knowing the real servo positions makes it impossible to reliably follow a preset gait. Lacking this prerequisite, it was not attempted to calculate a stable gait using FRI or ZMP analysis.

It will be shown that it is still possible to synthesise a stable walking pattern, but it will be very dependent on the hardware configuration, since every change to the acting forces (gravity or servo torques) will change the reached positions and thus the gait. If a better controlling mechanism could be applied to the servos, the gaits would not be as vulnerable to hardware changes and could be reused under changed cirumstances.

3.2 An uncontrolled walking gait

This section describes a gait that was implemented using Johnny Walker. Since only little feedback was available, we focussed on designing a gait that would be reasonably stable even without any sensory feedback, making it possible to improve this gait online by adding control.

The gait used is dynamic, so the robot will fall due to gravity during single-support phase (only one foot on the ground). In former trials a human walking motion was mimicked. The human walking cycle is governed by the single support phase: both feet are on the ground only for a very short period of time. Since this approach failed, the gait presented herein is further away from the human walking pattern, making it less elegant but more stable. Figure 3.2 shows the simplified walking gait.

The robot takes small steps of not more than 7 cm. After each step the robots body oscillates back and forth, an effect of the insufficient servo control. To prevent amplification of these oscillations, there is a delay of about 500 ms until the next step is attempted.

The diagram in figure 3.2 shows servo angles on a time scale. Shown are the angles for a right step, i.e. the right legs swings forward, the robot stands on its right left leg. Note that the commands sent to the servos do not define a smooth function, but include jumping from one position to another. Of course, the servos do not exactly follow these functions. Instead, the actual servo angles can be described by a a smooth but crude approximation of the preset function. However, the result is a very jerky motion, but it also is the only way to fully use the servo's power. Since each servo is internally controlled by a proportional controller, the torque applied by a servo is small when the difference between desired and actual position is small. The great differences are needed to achieve

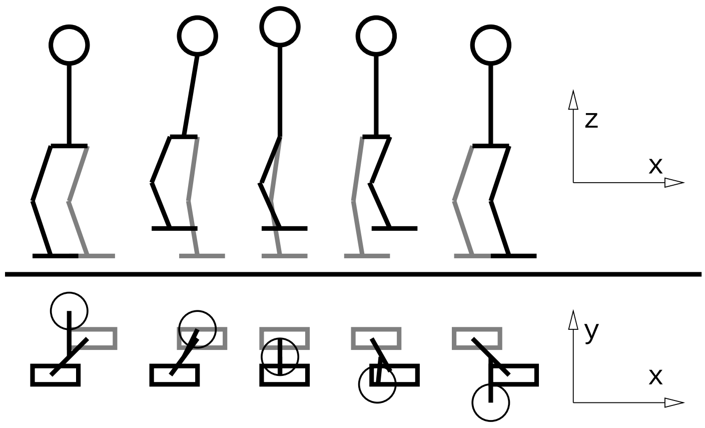


FIGURE 3.2: One half step in Johnny's walking gait. The hip joints are used to advance, not only to turn. The knee joints are bent backwards. Earlier experiments with a more human-like stance lead to similar but not quite as good results.

high torques, unfortunately the jerky motions promote the oscillations mentioned above. An alternative approach is described in section 3.3.

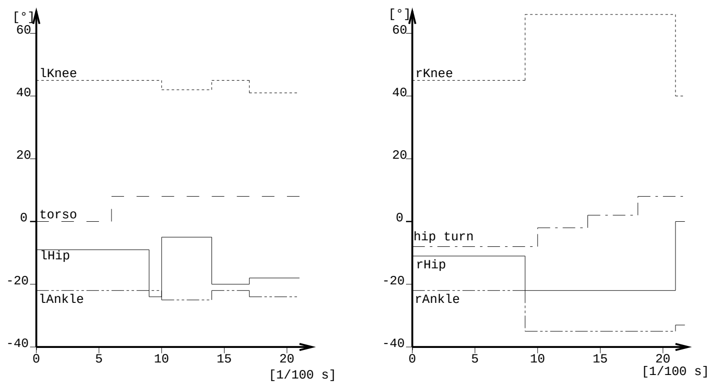


FIGURE 3.3: Servo angles for one half step in a time domain. The robot is standing on its left leg in this step. Shown are the angles the servos are set to, the actual servo angles remain unknown.

Unlike human walking, the advance made in each step is mostly based on hip turning rather than on moving one leg to the front. As can be seen in figures 3.2 and 3.2, the hip joints allowing the leg to swing back and forth are mainly used to lift the leg, whereas the forward motion is produced by turning the torso. This method has proven to be more stable than trying to advance bending the hip joints more. Also, the robot walks with its knees bent backwards, which we found easier to implement. Earlier experiments with a more human like posture lead to similar results, but the robot was slightly less stable.

To simplify further development, the gait is parametrised in terms of step length and step height as well as speed. Of course, this does not mean that every combination of values produces a working gait, e.g. taking very large, very fast steps will probably fail.

The robot can start and stop by itself, the gait does not have to modified to allow acceleration or deceleration. Walking is stable for ten to twenty steps, which is impressive considering that no feedback is used at all. Nevertheless, the robot will fall due to small errors in the gait or unexpected conditions (e.g. if the ground plane is not perfectly flat) that are not compensated online.

Theoretically, it would be possible to write a v, ω-Interface based on this walking gait. If the step length is set to different values on each side, the robot will turn. Unfortunately, the robot does sometimes slip, turning in a random direction even if it is meant to walk straight. To be able to correct this, feedback to determine the current robot's heading would be neccessary. A compass would be the easiest solution here. Outfitting the robot with rubber soles would improve the situation, but it could not entirely solve the problem.

3.2.1 Adding acceleration sensor feedback

The greatest problem preventing stable walking are the oscillations occuring after each step. Just waiting for a fixed amount of time does not solve the problem completely. The best solution would be to measure the current tilt angle directly after the step and then adjust the length and direction of the next step to balance the robot again. The acceleration sensors as well as the servos are not accurate enough to allow this kind of correction. Instead, we take series of samples from the acceleration sensors and calculate their variance until this drops below a threshold, indicating that the robot is standing still. Once this is the case, the next step is attempted.

Using this method, it is possible to walk as long a the surface allows it. One has to accept waiting times of up to a half a second between steps when walking without batteries (with external power supply) and up to two seconds with batteries carried. These times represent the duration of uncontrolled movement after each step. Since we cannot determine the exact type of movement (i.e. its direction), and we cannot correct it, the only way to deal with it is to wait until it subsides and the robot's state is known again.

3.3 A different approach to gait generation

Even though the robot walks with the method described, it is unsatisfactory in many ways:

- Most of the actual motion is unknown
  As already mentioned, the gait does not define a trajectory that is closely followed by the servos, instead it consists of only a few commands, assigning new destinations to a servo. How and when a servo arrives at the new position depends on the servo and the external forces acting at that time. The actual function desribing the servo

angle can be guessed but is unknown. It can not be measured since no posture feedback is available.

- Long waiting periods between steps
  Due to the jerky motion of the robot as well as the simple builtin position control in the servos, the robot's state is unknown after each step. Just to wait until we can measure the robot is standing still avoids the problem rather than solving it.

- Portability
  As soon as any of the robot's characteristics change, the gait will need manual readjustment. This process is based on trial and error, since changing one of the gait's features changes the gait in a hardly predictable way. A more systematic method would be desirable.

These problems might be resolved or at least improved if the robot's motions were smooth and completely known. Smooth motions would prevent or at least reduce oscillations. If the functions the joint angles follow would be known, one could analyse the gait and improve parts without implicitly modifying the rest of the gait as well.

In [1], a visualisation tool was used to manually enter angle values for each timeframe of 20 ms. These would be downloaded with the program and form the gait. This way of designing the gait is rather tedious, but it leads to smooth motions. It was also attempted to use functions to describe the robot's motion, these were evaluated while the gait was executed. Due to the limited processing power and the used trigonometric functions the update rate dropped below 12 Hz. This update frequency is insuffficent if smooth motions are required.

The first approach of manually designing a gait was not followed further. It does not allow parametrisation which makes online adaption impossible.

Using periodic functions to describe a gait is more promising as it allows to change parameters like step height and length without having to adjust all angle values. To increase the update frequency, the functions are evaluated before the robot starts to walk. The resulting servo angles are stored in memory and sent to the servos at an update rate of 50 Hz. This is the highest applicable frequency, since the PWM-generator used to synthesize the servo signal works with a 20 ms period.

To ensure reusability, classes were implemented, providing an object-oriented interface to the servos and to a gait. A documentation of these classes can be found in appendix A.

Due to a lack of processing power, it is not possible to evaluate the functions while executing the gait. Hence function parameters can not be changed while walking, but the gaits can be expressed in a fully parametrised form and parameters can be changed before each walking trial without having to compile or download the program anew.

The main problem with this approach is that the applied servo torque is small at all times, since the servos follow the preset trajectories quite accurately. If the difference gets greater due to external forces, the robot has already lost its balance. Typically, the supporting leg (i.e. its ankle) collapses at the beginning of the single-support phase, as soon as the trailing leg leaves the ground. It might be possible to create a gait that does not require high torques, but this was not achieved due to a lack of time.

Chapter 4

Visual Feedback for bipedal walking

The most prominent sensor on the EyeBot platform is the camera. On the robot in use this is a grey scale QuickCam ([1]).

Cameras are probably the sensors that can gather the most information. This information is encoded in an image or even a sequence of images. The computational cost to process images is high, limiting the amount of data that can be extracted from a camera image in real-time.

The information most important to us that could be obtained from the camera is the inclination angles of the robot's torso. To extract this data, the camera will take pictures of an artificial horizon. In this setup, the camera works as a three-axis inclinometer and can replace or complement the accelerometers in use.

In all the following calculation we will not deal with lens distortion, assuming that the camera is calibrated and the image is transformed to correct errors or all errors made due to lens distortion are small.

4.1 The artificial horizon

To measure the tilt angle of the camera, the robot looks at a screen as shown in figure 4.1. The screen is white with a black line on it. The line's position is chosen to be in the centre of the camera image when the robot stands upright. If the camera is rotated, the image changes and the position of the line in the picture reflects the rotation angle. Figure 4.2 shows images taken under different rotation angles in all directions.

The information that we will measure directly in the image is the horizontal and vertical displacement of the line and the line's dimensions (see figure 4.3 (a)). Using these values it is possible to approximate the three inclination values φ_x, φ_y and φ_z. Note that so far only rotation of the camera is considered.

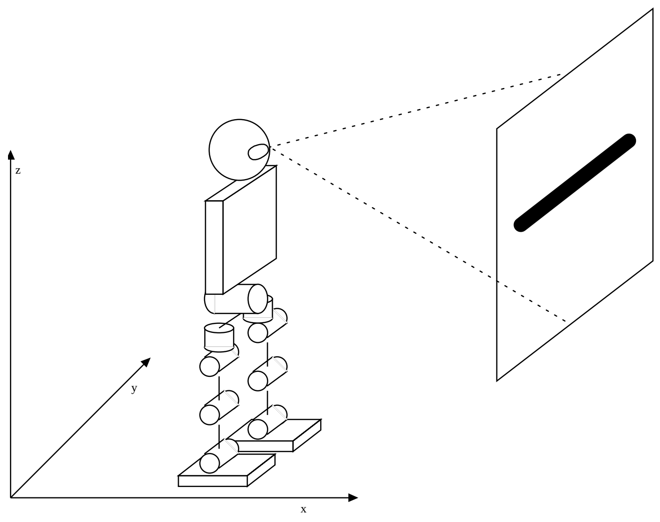


FIGURE 4.1: Artificial horizon experimental setup

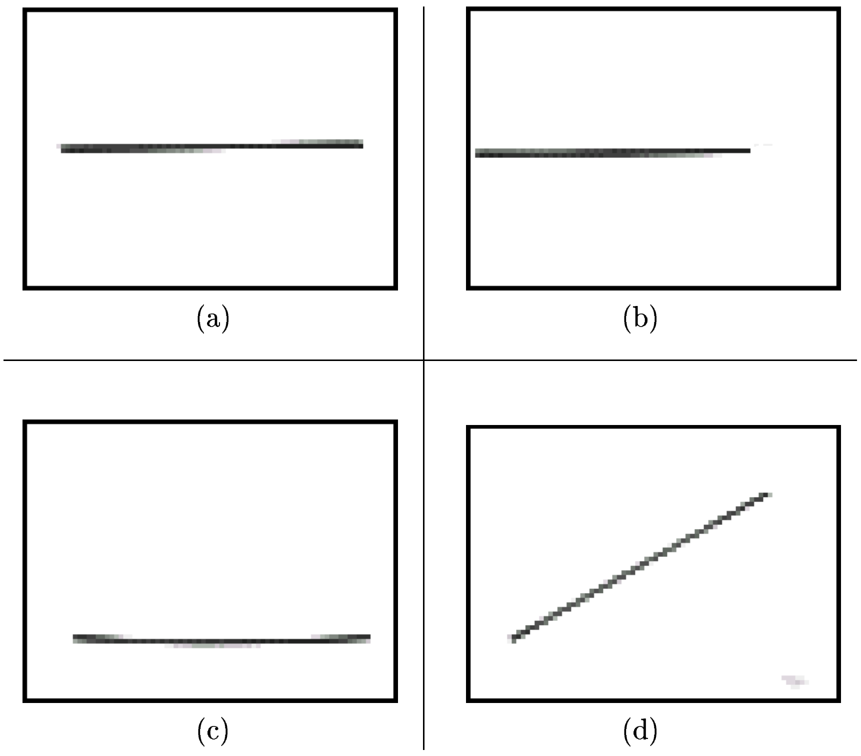


FIGURE 4.2: Pictures taken by the camera (a) while the robot is standing upright; with the camera rotated around the (b) z-axis, (c) y-axis and (d) x-axis.

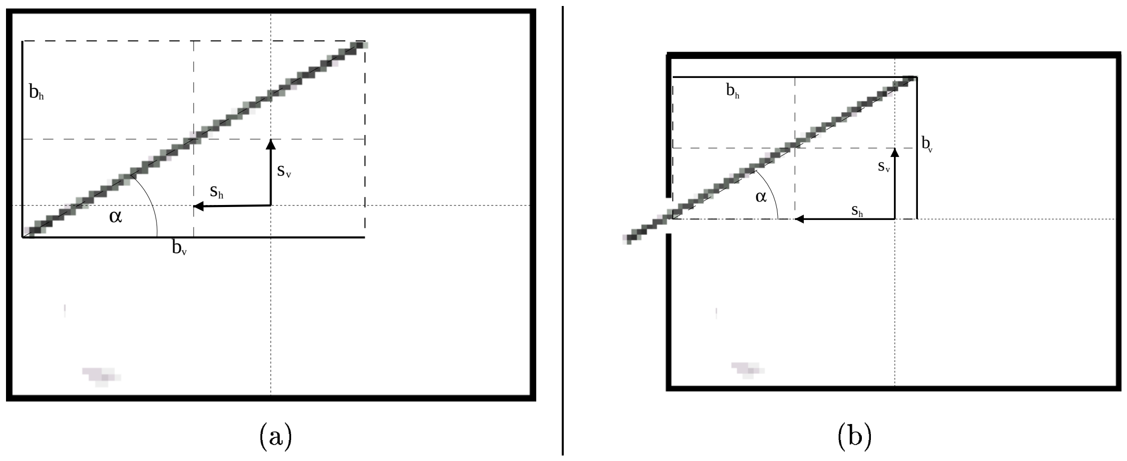


FIGURE 4.3: (a) Values obtained from a camera image: the horizontal and vertical line displacement s_h and s_v, the line's dimensions b_h and b_v as well as the tilt angle of the line in the image α.
(b) Values obtained if the line is only partly visible. The apparent values are incorrect, only the ratio b_v/b_h and with it $\tan|\alpha| = b_v/b_h$ remains the same.

4.1.1 Rotation around y- and z-axis

Figure 4.4 shows the details of the experimental setup with the camera being rotated around its y- or z-axis. These two rotations can be treated exactly the same and will not be discussed separately.

The rotation angle φ cannot be calculated directly, since the point C cannot be tracked in the image. φ is connected to the angles β_1 and β_2 as in equation (4.1).

$$\varphi = \frac{\beta_1 + \beta_2}{2} \tag{4.1}$$

The points A and B can be identified in the image, and thus the distances y and b can be measured. The angles β_1 and β_2 are determined by

$$\begin{aligned} \tan\beta_1 &= \frac{y}{d} \\ \tan\beta_2 &= \frac{y+b}{d} \end{aligned} \tag{4.2}$$

With equations (4.1) and (4.2) we obtain an expression for φ:

$$\varphi = \frac{\arctan\frac{y}{d} + \arctan\frac{y+b}{d}}{2} \tag{4.3}$$

The constant d is still unknown but will not be needed later. Because of the high computational requirements trigonometric functions should be avoided if a high processing rate is to be achieved. For small angles the tan and arctan functions can be approximated by

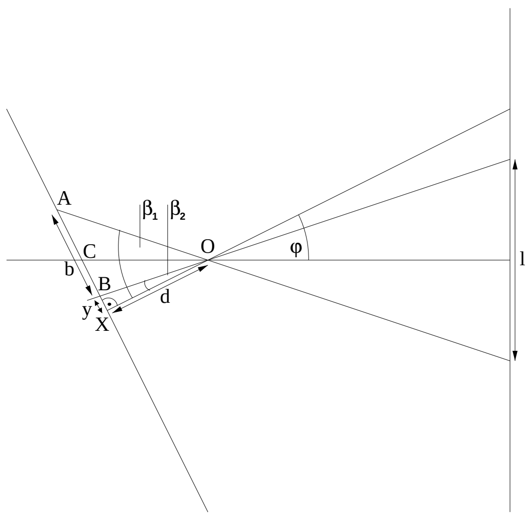


FIGURE 4.4: A view along the axis of rotation. Shown are the rotation angle φ, the end points of the line A and B, its centre in the image C, the image centre X as well as the angles $\angle AOX = \beta_1$ and $\angle BOX = \beta_2$ respectively; the length of the line's image $b = |\overline{AB}|$, its distance to the image centre $y = |\overline{AX}|$ and the internal camera constant d, as well as the length of the line in reality, l.

linear interpolations. In our case, these would be:

$$\begin{aligned} \tan\alpha &\approx \frac{R}{\Theta \cdot d} \cdot \alpha = k\alpha \\ \arctan\alpha &\approx \frac{1}{k} \cdot \alpha \end{aligned} \tag{4.4}$$

Where Θ is the camera's view angle and R is its resolution.

For the relatively small view angle of the camera used in our experiments (approx. 15°) the aproximation's absolute error $\Delta\varphi$ is always smaller than 0.15° — one pixel represents 0.375° (on the average) so this error is neglectable. As the view angle increases, the absolute and relative errors grow. If a camera with a considerably larger view angle would be used, a better approximation had to be found.

Using the approximation given by equation (4.4) in (4.3) finally gives a computable expression for φ:

$$\varphi \approx \frac{\frac{1}{k} \cdot \frac{y}{d} + \frac{1}{k} \cdot \frac{y+b}{d}}{2} = \frac{\Theta}{R} \cdot (y + \frac{b}{2}) \tag{4.5}$$

Note that $(y + \frac{b}{2})$ is just the distance from the image centre to the line centre, denoted s_v or s_h in figure 4.3.

4.1.2 Rotation around x-axis

These equations allow us to determine rotation angles around both y- and z-axis. As we will see, calculating the rotation of the camera around the view axis (x-axis) is more

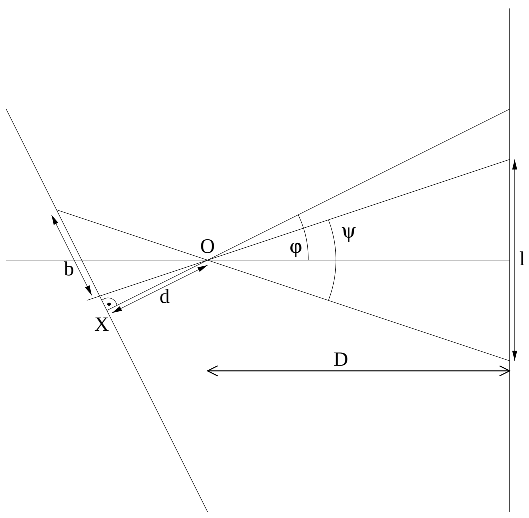


FIGURE 4.5: Another view along the axis of rotation. Shown are the length of the line in the image b, the length of the line in reality l, the angle covered by the line in the image ψ and the camera rotation angle φ.

difficult. We can easily calculate the tilt angle of the line in the image α (see figure 4.3):

$$\tan|\alpha| = \frac{b_v}{b_h} \tag{4.6}$$

Only the absolute value of α can be computed, since the line dimensions b_h and b_v are always positive. We should not apply the approximation given by equation (4.4) here, since α can be as big as 45°.

Unfortunately, this angle is not exactly the angle φ_x we are looking for. To show that it is a good approximation, we have to take a closer look on what happens to a line in different parts of the image.

In figure 4.5 we see the rotated camera again, this time focusing on the length of the line l and the length of its image b. When the camera rotates, the angle the line occupies in the field of view, ψ, remains constant, while b changes dependent on φ:

$$\tan\frac{\psi}{2} = \frac{l/2}{D} \tag{4.7}$$

$$b = d \cdot (\tan(\varphi + \psi/2) - \tan(\varphi - \psi/2)) \tag{4.8}$$

Applying (4.8) and substituting b_h and b_v in equation (4.6) leads to

$$\tan|\alpha| = \frac{b_v}{b_h} = \frac{\tan(\varphi_y + \psi_y/2) - \tan(\varphi_y - \psi_y/2)}{\tan(\varphi_z + \psi_z/2) - \tan(\varphi_z - \psi_z/2)} \tag{4.9}$$

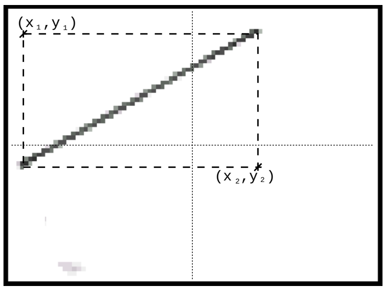


FIGURE 4.6: Coordinates extracted from the camera image

Using the approximation described in equation (4.4), we can transform (4.9) into:

$$\frac{b_v}{b_h} \approx \frac{\psi_y}{\psi_z} \tag{4.10}$$

Which is now an expression independent of φ. After applying (4.7) we can write equation (4.11).

$$\frac{b_v}{b_h} \approx \frac{\psi_y}{\psi_z} = \frac{2 \cdot \arctan\left(\frac{l_v/2}{D}\right)}{2 \cdot \arctan\left(\frac{l_h/2}{D}\right)} \approx \frac{l_v}{l_h} \tag{4.11}$$

Substituting l_v and l_h in (4.11) by expressions of l and φ_x leads to:

$$\tan|\alpha| = \frac{b_v}{b_h} \approx \frac{l_v}{l_h} = \frac{l \cdot \sin|\varphi_x|}{l \cdot \cos|\varphi_x|} = \tan|\varphi_x| \tag{4.12}$$

4.2 Implementation

Let us summarize the equations that determine φ_x, φ_y and φ_z:

$$\begin{aligned} |\varphi_x| &\approx \arctan\frac{b_v}{b_h} \\ \varphi_y &\approx \frac{\Theta_v}{R_v} \cdot s_v \\ \varphi_z &\approx \frac{\Theta_h}{R_h} \cdot s_h \end{aligned} \tag{4.13}$$

The horizontal and vertical view angles Θ_h and Θ_v as well as the horizontal and vertical image resolution R_h and R_v are known constants. The line displacement given by s_h and s_v respectively and the line width and line height b_h and b_v can be measured in the image. To do so, both end points of the line have to be located as their coordinates (x_1, y_1) and (x_2, y_2) are needed for the further calculations (see figure 4.2).

Knowing these coordinates, the necessary values can be computed:

$$\begin{aligned} b_h &= x_2 - x_1 \\ b_v &= y_2 - y_1 \\ s_h &= \frac{x_1 + x_2}{2} - \frac{R_h}{2} \\ s_v &= \frac{y_1 + y_2}{2} - \frac{R_v}{2} \end{aligned}$$

Where as usual in image processing, the point $(0, 0)$ is in the upper-left corner of the image.

To find the two coordinate pairs, the grey scale image taken by the camera was first converted to a black and white image. To prevent noise from falsifying the measurement, only pixels with two or more black neighbours are considered black. x_1 and x_2 are set to the x coordinate of the leftmost and rightmost black pixel respectively. Accordingly, y_1 equals the y-coordinate of the topmost black pixel whereas y_2 is the y-coordinate of the lowest black pixel.

As mentioned earlier, the calculation of b_h and b_v allows only positive values, to determine the sign of φ_x, the amount of black pixels in the top-left and top-right corners is compared. If the line starts in the top-left corner, φ_x is negative.

A C-library that implements the described functionality was written and its functions are described in appendix B.

Calculating tilt angles using equation (4.13) works well assumed that the whole line is visible in the image. If the turning angle around the y- or z-axis becomes too large, the line will partly leave the image (see figure 4.3 (b)). φ_x can still be calculated the same way it used to be, although accuracy is lost as the visible part of the line becomes smaller. The ratio b_v/b_h will remain the same even if only a part of the line is visible.

Both s_v and s_h can not be measured any more. If the size of the line's image would be known, it would be possible to calculate the virtual line's centre. One could also mark the line's centre D so it could be tracked directly without involving measurement of the end points A and B. Without this adjustment, the maximum measurable y- or z-angle is limited even beyond the camera's view angle.

4.2.1 Calibration

Assuming the line to be in the image centre is not satisfactory. Most of the time one will not succeed in setting up the experiment accurate enough. The result is a slight displacement of the camera in either y- or z-direction or the camera will be slightly tilted and does not exactly face the screen. In this case, the angles φ_x, φ_y and φ_z are non-zero. A calibration routine was added that takes a picture and assumes that the angles extracted from this picture are the angles in zero position. They will be subtracted from all further measurements. Note that if the non-zero angles detected by the calibration function are

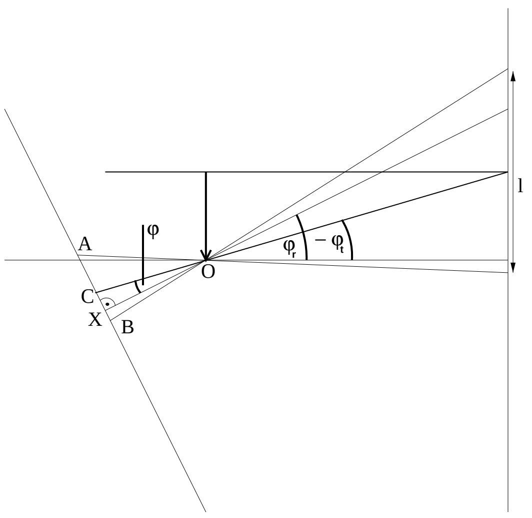


FIGURE 4.7: The camera is rotated as well as displaced. The apparent rotation angle φ consists of the truly rotational part φ_r and the angle φ_t originating from the displacement of the camera

caused by a camera displacement, they are dependent on the distance from the screen and calibration has to be repeated whenever the distance to the screen changes.

4.3 Camera displacement

Of course, it is unlikely that the camera only rotates and does not change its position if used on a robot. Any camera displacement will cause a change in the angle φ_y or φ_z. Figure 4.3 shows a camera that is rotated as well as displaced. The translation will be perceived as a rotation as well — the rotation angle φ is split in two parts: the rotational part φ_r and the angle caused by the camera displacement, φ_t. Unfortunately, using the approach presented here, it is not possible to separate the translational and rotational parts of the line position.

The effects of the translation on the line size in the image are about as important as the effects that were neglected when using the approximation (4.4). This is an advantage because the formulae used to determine the tilt angles can still be used even if translation is considered. In this case, $\varphi = \varphi_t + \varphi_r$ is measured, not only the rotational part φ_r.

When the camera is used as feedback for a control algorithm that keeps the robots walking towards the screen, it is not necessary to distinguish between translation and rotation. In both cases, the robot will have to correct its heading towards the direction in which the line is located in the image. Also when applying the sensor to balancing as done in section 4.4, no translational feedback is needed.

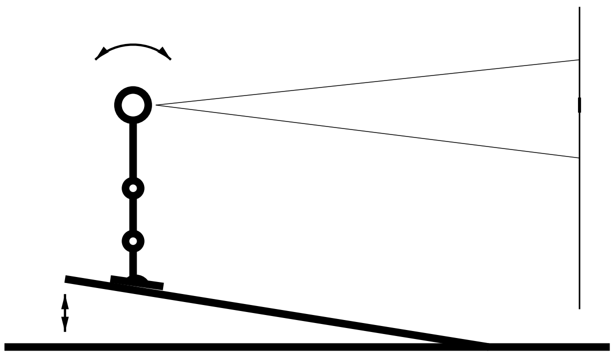


FIGURE 4.8: Balancing experiment using the camera to determine the robot's tilt angle. The surface the robot is standing on can be tilted; the robot has to balance by adjusting the angle of its ankle joints.

4.4 Application

In order to verify the applicability of the visual inclinometer, it was used in balancing. In the experiments conducted, the robot "Johnny Walker" stands in front of the artificial horizon, on a surface that can be tilted around the robot's y-axis (see figure 4.8). The tilt angle around the y-axis is measured using the camera and used in a simple P-controller to adjust the robot's ankles and make it stand upright. The program can read 25 values a second from the tilt sensor, limited by the camera's frame rate.

The same experiment was conducted using the acceleration sensors. Due to the servo vibrations, their readings are quite noisy. If an acceleration sensor reading causes a major servo action, the high current causes a voltage fluctuation that disturbs the next measurement. If this is the case, the robot will almost certainly loose balance. To avoid this, the surface the robot is standing on has to be held very stable and the tilt angle cannot be changed as fast as it is possible using the camera.

It was also attempted to use the camera to walk straight using the gait described in section 3.2. The gait turns the robot's torso around its z-axis to advance. Unfortunately, the camera turns so far that the line will leave the image. In this configuration, the visual inclinometer is not applicable. Either the robot's step length has to be significantly decreased or a camera with a wider view angle has to be used.

4.5 Acceleration sensor vs. camera

Neither the acceleration sensor nor the camera is clearly superior. Which sensor is best to use strongly depends on the situation. If a sensible logic which selects the appropriate sensor to use can be found, a combination of both is probably best.

The camera definitely is the more reliable sensor. Its readings are stable and not subject to any kind of noise. The accuracy is better than one degree — far better than the acceleration sensor, especially if the servos are working. The camera does not depend on a stable reference voltage, which cannot be guaranteed when the servos are moving. Thus, correct readings can be obtained at all times, not only when the servos are not moving. If the robot's torso is moving during a gait, other accelerations than gravity act on it and so the values obtained from the acceleteration sensors do not reflect the torso's tilt angle.

Of course, if used in the right way, the acceleration sensor readings contain more information than only the direction down. In combination with a reliable inclinometer they can give valuable information about the robot's movements.

The greatest problem when using the acceleration sensors is their poor accuracy and the extremely noisy values. An advantage is that acceleration sensor readings are always available. To use the camera, an artificial horizon has to be set up. The characteristics of the setup used in the experiments herein limit the area in which the sensor can be used: It will only give values in a small area directly in front of the screen. The small view angle of the camera further resticts the area of application.

The visual inclinometer cannot distinguish between translation and rotation of the camera. Depending on the application, this can be a major drawback or an advantage. For balancing, pure rotational (tilt) angles would be needed, but the combined readings can still be used.

Chapter 5

Conclusion

The work done leads to a deeper understanding of the walking problem in general and the specific problems arising on the “Johnny Walker” platform. Walking was finally implemented on the robot. The major hindrances for further development were identified and possible solutions could be offered.

In chapter 2, the theoretical background of walking was presented. The confusion around the ZMP was cleared up as already done in [8]. A comprehensive discussion of the important tools for gait analysis was given. Their use for gait analysis, gait generation and control was examined. As in so many cases, the utility of a tool and the cost for its prerequisites are closely correlated.

The walking gait designed for “Johnny Walker” was presented in chapter 3. The first stage of development was to implement an almost stable walking pattern that would only require minimal online adjustments. After this was achieved, basic feedback from the acceleration sensors was added to stabilize the gait.

The resulting walking gait enables the robot to walk on an even ground plane as long as the battery and the ground plane permits it. It is parametrised in terms of step length, step height and speed, which allows to set different step lengths on each side and thus turn the robot. This feature can be used to write a simple v, ω-interface, to do so additional feedback on the robots heading is required, e.g. using a compass.

Unfortunately, the gait is not described as a set of smooth functions, but as a set of servo commands. This means that the actual movements of the robot are unknown as they cannot be measured without posture feedback. This makes gait analysis impossible. The gait is also extremely vulnerable to hardware changes. An alternative method of specifying a gait was presented and classes were implemented to allow easy access to it.

The acceleration sensors are used to determine the direction of gravity and thus the torso inclination. Experiments showed that the sensors are too noisy to be used for direct feedback. In the method described in section 3.2.1, only the variance of subsequent samples was taken into account, not the sample values.

To offer an alternative to the acceleration sensors, a visual inclinometer was designed and implemented. Because image processing has high computational requirements, the task had to be made as simple as possible. An artificial horizon was set up and a single black line was tracked in the camera image, allowing a frame rate of 25 processed images per second. According to the line's position and tilt, the three rotation angles of the robot's torso could be extracted. In terms of accuracy, this method is far better than the acceleration sensors. It could be succesfully applied to a balancing problem.

There are major problems with this approach. The area in which it can be used (the robot has to face an artificial horizon) is severly restricted. The camera's view angle is very small, so are the maximum detectable rotation angles. The rotation angles occuring during normal straight walking are already greater than these limits, so the visual inclinometer could not be succesfully applied to walking.

Chapter 6

Future work

What has been done is of course not complete and requires more work to investigate unresolved or newly raised questions.

6.1 Work on "Johnny Walker"

The gait synthesized for the robot is a good experimental basis to test new sensor equipment and determine its use. It is almost stable even without feedback so new sensors can be quite easily incorporated and compared.

The first sensor to be considered are foot force sensors. They would allow ZMP measurement. This data can be used for gait analysis, which could not be done so far since no suitable feedback data was available. A gyroscope or similar inclination sensor can replace or complement the acceleration sensors and provide a more accurate tilt measurement. Especially here, the utility of the feedback obtained is to be weighed up against the cost of the equipment considering that the robot is meant to be a low-cost solution.

Posture feedback can be used in many useful ways. First of all, it would allow us to calculate the robot's CoG. Most probably, this can not be done online due to the limited processing power, making a CoG trajectory control impossible. Still, gaits can be designed and manually adjusted using the sensor data from former trials.

This kind of feedback can also be used to implement a different (more advanced) position control for the servos. It is not possible to seperate the motor controller and the internal feedback mechanism in the servo. Since the EyeBot controller cannot replace the built-in motor controllers for that many servos, it is not possible to disable the built-in position control. Thus, a control logic has to be implemented on top of the existing controller, making it hard to determine stable control parameters. Torque control is also possible using the same approach. If this could be accomplished, a generic servo can be used the same way a DC-motor is used, giving the software (gait) designer a lot more freedom.

Since the servos have internal position feedback, this data could be reused. There is hardly enough space on the robot to mount extra potentiometers or other sensors that

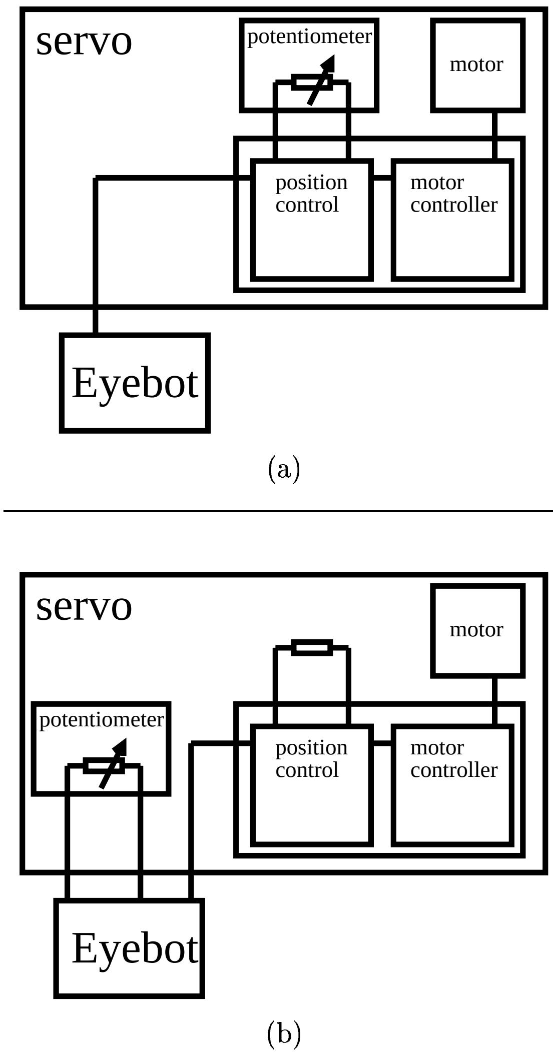


FIGURE 6.1: Possible Modifications to a servo:
(a) the original servo contol loop
(b) giving modified potentiometer readings

allow position feedback — using the servos own potentiometer could solve this problem.

Figure 6.1 (a) shows a schematic drawing of the control loop in the modified servo. The servo has a built-in potentiometer, allowing it to control its position. The motor is conrolled by a motor controller and a feedback loop using the feedback data obtained from the internal sensor.

One can modify the servo to disable the internal control algorithm. To do so, the potentiometer of the servo controller is replaced by a resistor. Its resistance should be the potentiometer's resistance in neutral position (as shown in figure 6.1 (b)).

Thus, the servo's internal control mechanism behaves just like a motor controller: If the servo is set to neutral position, the controller will stop the motor, since the bogus potentiometer reading indicates the servo is in neutral position. If the servo position is set to any other value, the controller will start the motor in the appropriate direction, with a torque that is proportional to the difference between the given and the neutral position.

Given the possibility to control the motor's torque, one can easily implement a feedback loop of any kind in software, using the feedback obtained from the servo's potentiometer. For each servo to be modified, one analog input has to be available on the controller and one additional wire is needed to feed the potentiometer readings into the controller. Note that the implemented controller has to take care not to break the servo by turning it further than possible and should reduce torque when coming closer to the extreme positions.

With or without additional feedback, the gait designed can be improved further or used in a future project. The most obvious application would be to attempt writing a v, ω-interface for the robot. Therefore, the robot should be outfitted with rubber soles to prevent it from sliding. Additional feedback on its heading, e.g. using a compass is probably needed as well. Using this gait, the possible values for v and ω would be very limited, the robot's velocity might not be controllable at all due to the unknown waiting times between individual steps.

The alternative gait generation and execution method presented in section 3.3 is worth following further. It allows to synthesize well-defined gaits. If additional feedback is available, these gaits can be analysed and adjusted online. I am convinced that it is possible to synthesise a gait that is as stable as the one presented using smooth movements. Preferably, this gait will be more human-like than the existing one.

6.2 Building a new robot

To take full advantage of the experiences gained in the experiments done with "Johnny Walker", a new robot has to be built. Starting from the current design, major modifications should be:

- Weight reduction
  The 3 mm aluminium brackets used in the construction of "Johnny Walker" are definitely oversized. Just like in nature, the robot's joints are its weakest part. It is not necessary to make its segments much stronger than its joints, especially if this strength has to be paid for in weight. A structure made of plastic or limbs only consisting of servos could save weight and unburden the servos. The inertia forces acting will also be smaller, making is easier to change the direction of movement for any part of the robot. The lighter construction will also make it possible to build the robot even smaller.

- Additional joints
  I suggest two additional degrees of freedom in each leg - enabling the hip as well as the ankle joints to rotate around the x-axis. This can be used to produce an angular momentum around the x-axis which so far could only be done by moving the torso. The legs could also be used to compensate for unbalanced momentum by stepping sideways.

- Combined joints

  Some of the robot's joints rarely move independently. The hip, knee and ankle joints are always used to lift the foot, keeping the foot paralell to the ground. Instead of using three degrees of freedom, this can also be done using just one servo and moving the joints via tendons. This would simplify the gait as only one servo had to be controlled (lift/straighten leg) and it would save additional weight.

- Arms

  Arms are a good way to shift mass forth and back. They can be used to counteract small moments by moving the robot's CoG or produce a moment around the z-axis that can compensate for other movements.

  To make arms useful, they should have at least two degrees of freedom each. One to swing the arm forth and back and one to move it outwards. The latter can be used to shift the robot's CoG along the y-axis and it ensures that the arms are usable even when the torso is tilted to one side.

  Of course, arms add to the weight and cost of the robot, so it has to be carefully examined if they are needed and wanted.

- Additional sensors

  A sensor that could be realised easily would be posture feedback. The design of the new robot should leave enough room for potentiometers in the robot's joints. The data obtained describing the robot's posture can be used to implement torque and position control for the servos used (as described in section 6.1). Using external potentiometers, the servos do not have to be modified in order to retrieve the required information.

  In combination with sensors capable of detecting foot-ground contact, posture feedback can also be used as a torso inclinometer (see section 2.6). From the experiences made herein we know that the servos do not reliably reach their assigned positions. Without any position control available, it will be hard to do any kind of gait analysis, since the actual servo positions and with them the actual gait remains unknown.

  In order to conduct ZMP analysis on the new robot, it has to be outfitted with foot force sensors. Having universal force-moment sensors is not an option for a robot of this size. One could probably construct sensors that could be used on this robot — however the money and effort doing this are most probably better spent in other parts of the design.

- Controller

  The current EyeBot controller was sufficent for this work. It came to its limits when trigonometrical functions had to be evaluated in real-time (see section 3.3 or [1]).

  Once more sensory feedback has to be processed, e.g. to implement control for the actuators, the computational requirements will grow beyond what can be provided by the current EyeBot controller. Especially if more image processing is done, the computational power available will limit the accuracy and reliability of the application.

Choosing the wrong controller for the new robot would not cause serious problems since the controller is a part of the robot that can be easily exchanged without changing the hardware design.

Whatever design is chosen for the new robot, the parts used should be generic components as far as possible. This ensures easy maintenance since a defective component can be replaced if neccessary. On the other hand, generic components often have to be replaced when broken, where a custom-built component would be repairable.

In terms of reliability the generic components are also likely to be better than previously untested parts. There are only few parts of the robot that are worth being individually developed, mostly where no generic solution is available. Universal force/moment sensors in a size suitable for deployment on this robot are one example.

6.3 Work on visual sensors

Implementing a visual inclinometer as it was done can only be the first step towards using the camera as a sensor and incorporating its feedback into the control algorithms.

Using a wider view angle, the sensor could probably be applied to a broader range of problems, including walking. Trying to make the robot walk in a straight line towards the screen would be the next step to be taken.

Since an artificial horizon can not be available at all times, it would be desirable to be able to process real-world images. An approach similar to the one used would be to identify lines that should be horizontal (or vertical) and calculating their tilt angles. Especially in aritficial environments like buildings, there are a lot of these lines. If one line has been identified in the image, it is possible to use this line as we did with the artificial horizon, determining y- and z-rotation angles relative to the angle at which the line was first seen.

Another method could use optical flow methods to determine the robot's movement. The processing power needed for both of these tasks is far above what is available on the current EyeBot. In order to apply these methods or extend the use of the camera in a different way, a stronger processor or an additional processor is needed.

One could also think about using a seperate computer for image processing, possibly one not carried by the robot itself. The image data would have to be transferred to the processing unit, if possible using wireless communication. However, using a computer not carried by the robot would make the robot non-autonomous. In future, the technology for onboard image processing will become feasible.

Appendix A

CServo and CGait classes

A.1 CServo

The **CServo** class wraps around one servo to hide its hardware specific characteristics and offers an object oriented interface to it.

Constructors

```
CServo();
CServo(const int hdt_sem, const bool autoapply = false);
CServo(const int hdt_sem,
       const int minangle,
       const int maxangle,
       const bool autoapply = false);
CServo(const CServo& from);
```

If the object is constructed without supplying a semantic constant, `valid()` will return `false`. The optional `autoapply` parameter determines if changes to the servo's position are automatically applied or have to be confirmed using the `apply()` method.

`minangle` and `maxangle` are used to compute a servo position value from an angle. If these parameters are given, it is possible to send angles to the **CServo** object.

Setting the minimum and maximum angles

```
bool setAngles(const int minAngle, const int maxAngle);
int minAngle() const;
int maxAngle() const;
```

These functions set and read the values that are used to convert angles to servo positions.

Setting the servo position

```
bool set(const int pos);
bool setAngle(const int angle);
int pos() const;
int angle() const;
```

These functions set the servo position. The `set(...)` and `setAngle(...)` return `true` on success. In order to use `angle()` and `setangle()`, the minimum and maximum angles have to be set. If the object is set to automatically apply the changes, the servo will immediately change its position or wait for a call to `apply()`.

Automatically applying changes

```
bool apply();
void setAutoApply(const bool autoapply);
bool autoApply() const;
```

Using `setAutoApply(...)`, the object can be set to automatically applying changes made to the servo position. If this is the case, `apply()` has no effect. `autoApply()` return the current state.

Miscellaneous

```
int semantics() const;
bool valid() const;
```

These functions return the semanantic constant the object was created with and if this constant is zero respectively.

A.2 CServos

The **CServos** class auto-detects all servos in the HDT-table and instantiates them as **CServo** objects. They are accessible by their semantic constants.

Constructor

```
CServos(const bool autoapply = false);
```

The `autoapply` parameter is passed on to the constructors of the **CServo** objects. For all servos in the HDT-table, a **CServo** object will be created.

Accessing servos

```
CServo& operator[](int semantics);
const CServo& operator[](int semantics) const;
```

These functions grant access to a servo specified by its semantic constant.

Automatically applying changes

```
bool apply();
void setAutoApply(const bool autoapply);
bool autoApply() const;
```

These functions act the same way as their **CServo** counterparts. In fact, a call to one of these functions is directly passed on to all servos on the system.

Miscellaneous

```
void midPos();
int count() const;
```

The `midPos()` function sets all servos to neutral position (128). `count()` return the number of servos installed.

A.3 CGait

An object of class **CGait** stores movement commands for all available servos (as recognised by a **CServos** object). The commands are composed from basic **Movement**s.

Constructor

```
CGait(CServos& servos, const int duration);
```

The constructor allocates enough memory to hold commands for all `servos` and initialises the gait to hold middle position for `duration`. Duration is given in 1/100 seconds.

Changing the gait's duration

```
int duration() const;
void setDuration(const int duration);
```

The `setDuration(...)` method shortens or prolongs the gait. Extra time will be filled with mid-pos commands. `duration()` returns the current duration.

Adding Movements

```
bool moveJoint(const int semantics, const Movement& movement);
```

This function adds the given **Movement** to servo specified by `semantics`.

Executing the gait

```
void execute();
```

This function executes the stored gait.

Displaying the gait

```
void show();
```

The gait is displayed on the EyeBot display. The user can scroll through the gait for debugging purposes. Displayed are servo control values i.e. values ranging from 0 to 255.

A.4 Movement

The **Movement** class is a base class for basic movement commands that can be sent to servos. They can be easily combined to allow creation of more complex actions.

To create a useful sub-class of **Movement**, only the protected method `f(int)` has to be implemented. It has to return an angle. During construction, this method will be called for each value between `start()` and `end()`. The resulting values are stored internally so that no calculations have to take place during run-time. To allow parametrisation of the new sub-class, useful constructors have to be supplied.

The **Movement** class offers many operators that allow to add or substract **Movement**s, or shift their start- and end-times.

Appendix B

VISINC library

This appendix describes the functions and data structures in the VISINC library.

B.1 Data structures

VISINCRegion

Declaration:

```
typedef struct
{
  int top, bot, right, left;
} VISINCRegion;
```

Purpose:

This structure stores the calculated coordinates in the **VISINCImage** structure.

VISINCImage

Declaration:

```
typedef struct
{
  /* pointer to a greyscale image */
  image *img;

  /* flags which values are already computed */
  int ValuesSet;

  /* internally used values */
  VISINC_Region *region;
  int XCenter, YCenter;
  float RotAngle;
```

```
    /* computed values (if set, refer to ValuesSet) */
    float XAngle, YAngle, ZAngle;
  } VISINCImage;
```

Purpose:
Stores all information available on one image. This prevents values from being calculated twice.

B.2 Functions

VISCINCGetImage

Declaration:

```
VISINCImage* VISINCGetImage(VISINCImage *img);
```

Description:
This function will take an image. If `NULL` is passed, a new **VISINCImage** structure will be allocated and a pointer to it will be returned. If a valid **VISINCImage** structure is provided, but no image is contained (`img->img == NULL`) a new image will be allocated. An image contained in the img structure will be overwritten and the data fields will be initialized with -1 (`XCenter`, `YCenter`) and 0 respectively.

The function returns `NULL` when an error occurs, a pointer to the **VISINCImage** structure the image was written to otherwise.

Reasons for failure are:

- an invalid, non `NULL` pointer is passed
- not enough free memory
- **CAMGetFrame** fails

VISINCFreeImage

Declaration:

```
void VISINCFreeImage(VISINCImage *img);
```

Description:
This function frees a **VISINCImage** structure and the contained image (if not `NULL`).

Reasons for failure are:

- an invalid, non NULL pointer is passed

VISINCCalibrate

Declaration:

```
int VISINCCalibrate(VISINCImage *img,
                    float XViewAngle,
                    float YViewAngle,
                    int Threshold);
```

Description:
Calibrates the zero position using img as reference and sets the b/w recognition threshold. If `NULL` is passed, a new image will be taken (with the new threshold). If the structure `img` points to contains precomputed data, this data will be used. If the structure `img` points to doesn't contain an image and no information is stored elsewhere, a new image will be taken.

The `XViewAngle` and `YViewAngle` parameters will be stored in globals, and will be used as the view angle of the camera. No changes will be made to the globals if an error occured.

The function returns 0 on success, `VISINC_ERROR` otherwise. It calls **VISINCGetImage**, **VISINCGet(X|Y|Z)Angle** and **VISINCFreeImage**.

Reasons for failure are:

- an invalid, non `NULL` pointer is passed
- one of the called functions fails

VISINCGet(X|Y|Z)Angle

Declaration:

```
float VISINCGetXAngle(VISINCImage *img);
float VISINCGetYAngle(VISINCImage *img);
float VISINCGetZAngle(VISINCImage *img);
```

Description:
These functions will extract the tilt angle in one direction from the image, considering the calibration. The raw information will be stored in the `XAngle`, `YAngle` or `ZAngle` fields of the structure referenced by `img`. One of the flags in `ValuesSet` will be set accordingly. If this is already the case, the stored value will be used for further calculation. The computed or stored value will be adjusted using the calibration value and returned.

The `img` structure will be left unchanged if an error occured. The function returns the angle on success, or `VISINC_ERROR` otherwise.

Reasons for failure are:

- an invalid or `NULL` pointer is passed

Bibliography

[1] Elliot Nicholls. Bipedal dynamic walking in robots. Honours dissertation, Department of Electrical Engineering, University of Western Australia, October 1998.

[2] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Turo Takenaka. The development of Honda humanoid robot. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1321–1326, 1998.

[3] Honda humanoid robot site. WWW site: http://www.honda.co.jp/robot/, 2000.

[4] The Shadow biped. WWW Site: http://www.shadow.org.uk/projects/biped.shtml, 2000.

[5] C. Chevallereau and P. Sardain. Design and actuation optimization of a 4 axes biped robot for walking and running. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 4, pages 3365–3370, 2000.

[6] M. Gienger, K. Löffler, and F. Pfeiffer. A biped robot that jogs. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 4, pages 3334–3339, 2000.

[7] M. Vukobratovic and D. Juricic. Contributions to the synthesis of biped gait. *IEEE Transactions on Biomedical Engineering*, 16, 1969.

[8] Ambarish Goswami. Postural stability of biped robots and the foot rotation indicator (FRI) point. *International Journal of Robotics Research*, 18(6), 1999.

[9] Qinghua Li, Atsuo Takanishi, and Ichiro Kato. A biped walking robot having a ZMP measurement system using universal force-moment sensors. In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, volume 3, pages 1568–1573, 1991.

[10] C. L. Shih, Y. Z. Li, S. Churng, T. T. Lee, and W. A. Gruver. Trajectory synthesis and physical admissibility for a biped robot during the single-support phase. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, volume 3, pages 1646–1652, 1990.

[11] Chih-Long Shih. Analysis of the dynamics of a biped robot with seven degrees of freedom. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 4, pages 3008–3013, 1996.

[12] Qiang Huang, Kenji Kaneko, Kazuhito Yokoi, Shuuji Kajita, Tetsuo Kotoku, Noriho Koyachi, Hirohiko Arai, Nobuaki Imamura, Kiyoshi Komoriya, and Kazuo Tanie. Balance control of a biped robot combining off-line pattern with real-time modification. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 4, pages 3346–3352, 2000.

[13] Anirvan Dasgupta and Yoshihiko Nakamura. Making feasible walking motion of humanoid robots from human motion capture data. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 2, pages 1044–1049, 1999.

[14] Chee-Meng Chew and Gill A. Pratt. A general control architecture for dynamic bipedal walking. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 4, pages 3990–3996, 2000.

[15] Qinghua Li, Atsuo Takanishi, and Ichiro Kato. Learning of robot biped walking with the cooperation of a human. In *Proceedings of the* 2^{nd} *IEEE International Workshop on Robot and Human Communication*, pages 393–397, 1993.

[16] Samuel Agus Setiawan, Jin'ichi Jamaguchi, Sang Ho Hyon, and Atsuo Takanishi. Physical interaction between human and a bipedal humanoid robot. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 1, pages 361–367, 1999.

[17] Jesse Pepper. Walking algorithms for biped robot. Honours dissertation, Department of Electrical Engineering, University of Western Australia, October 1999.