Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2357

Adaptive locomotion of snake-like robots

Florian Leiß

Course of Study:	Diplom Informatik
Examiner:	Prof. Dr. rer. nat. habil. Paul Levi
Supervisor:	Dipl.-Ing. Eugen Meister
Commenced:	06.06.2011
Completed:	06.12.2011
CR-Classification:	I.2.2, I.2.9, I.2.11

Abstract

Adaptive locomotion for multi-robot organisms is a huge challenge in modern robotics. The complexity grows fast with the number of degrees of freedom (DOFs) of a robot. A snake like organism structure usually consists of many DOFs and is hence restricted in their motion capabilities. In this theses, an adaptive gait algorithm for avoiding of obstacles should be analyzed by using techniques of coupled oscillators. The simulation in MATLAB as well as tests on a real snake robot will be done in this work.

In the MATLAB simulation we could see the principle of adaptive snake-like locomotion as it should be. As well we could analyse fitting functions for the oszillator in the Central Pattern Generator. On the real snake robot we had to take the theoretical foundations and adapt them to make them work on real-time operating system. It was faszinating to work on all difficulties be communication, configuration, designing or programming. And finally we saw the robot crawl arround obstacles, controlled by the equations we developed. It would be great to see our work live on in projects like SYMBRION and REPLICATOR.

Contents

List of Figures

List of Tables

1 Introduction

In modern robotics terrestrial locomotion is a huge challenge. In rough terrain a robot must find a possibility to get by or over any obstacle. In addition changing subsoils will make it difficult to move at all. A snake is one of the oldest still living life forms on earth. She lives as far northward as the Arctic Circle in Scandinavia and southward through Australia and Tasmania, in the sea and as high as 4,900 m in Himalayan Mountains of Asia.

Figure 1.1: Multi-robot organisms.

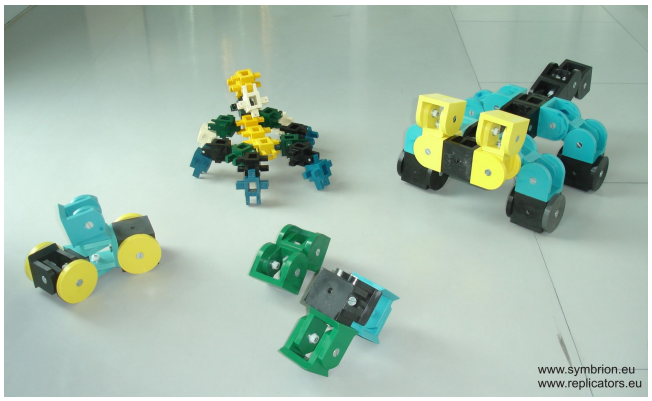


Figure 1.2: More Multi-robot organisms.

1.1 Motivation

Nature has found different great solutions for this task. There is legged locomotion with two, four, six or eight appendages, rolling and limbless locomotion. Every legged locomotion makes high demands on the sense of balance and the motor skill of the organism. Rolling is only seen on a small number of animals. Mostly they coil themselfs up into a ball to roll down hill. Limbless locomotion is used by molluscs such as slugs and snails, which use a layer of mucus that is secreted from their underside to move, and snakes. Most snakes use lateral undulation for their movement. In this mode, the posteriorly moving waves push against contact points in the environment, such as rocks, twigs, irregularities in the soil, etc. [Cogger, 1991] This thesis deals with snake-like robots so we will later take a closer look on the locomotion of the snakes.

Figure 1.3: A snake between rocks.

A snake robot could deal with difficult terrain just like a snake can trail, swim or even climb on trees. For search and rescue operations like in the RoboCup Rescue Robot League or in real emergency situations the robot could go on recon missions and find passages where no human could ever pass.

1.2 Former work

As we have seen in the former sections, nature has formed a flexible robust organism. So the IPVS build a more simple snake-like robot to discover the possibilities in snake-like locomotion without the challenges of connected swarm-robots. [Eugen Meister, 2011] The robot has 25 Degrees of freedom, 12 horizontally and 13 vertikally. In the beginning of this work there were approaches to move the snake in three dimensions, but the motors weren't strong enough to lift the central segments of the snake robot. So we set the

Figure 1.4: A snake straightening up.

horizontal motors to a fixed position and used only the 13 remaining motors. The characteristics of the robot are described in chapter 3.

1.3 Projects SYMBRION and REPLICATOR

In the SYMBRION (Symbiotic Evolutionary Robot Organisms) and REPLICATOR (Robotic Evolutionary Self-Programming and Self-Assembling Organisms) project the main focus is to investigate and develop novel principles of adaptation and evolution for symbiotic multi-robot organisms based on bio-inspired approaches and modern computing paradigms. Such robot organisms consist of super-large-scale swarms of robots, which can dock with each other and symbiotically share energy and computational resources within a single artificial-life-form. When it is advantageous to do so, these swarm robots can dynamically aggregate into one or many symbiotic organisms and collectively interact with the physical world via a variety of sensors and actuators. The bio-inspired evolutionary paradigms combined with robot embodiment and swarm-emergent phenomena, enable the organisms to autonomously manage their own hardware and software organization. In this way, artificial robotic organisms become self-configuring, self-healing, self-optimizing and self-protecting from both hardware and software perspectives. This leads not only to extremely adaptive, evolve-able and scalable robotic systems, but also enables robot organisms to reprogram themselves without human supervision and for new, previously unforeseen, functionality to emerge. In addition, different symbiotic organisms may co-evolve and cooperate with each other and with their environment. [SYMBRION, 2008]

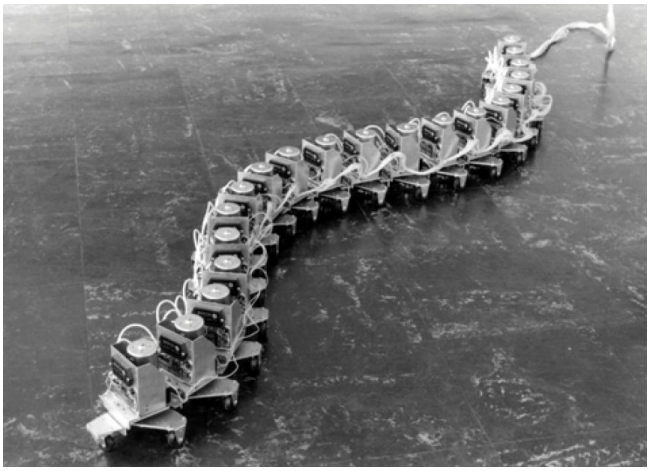


Figure 1.5: Hirose ACM prototype

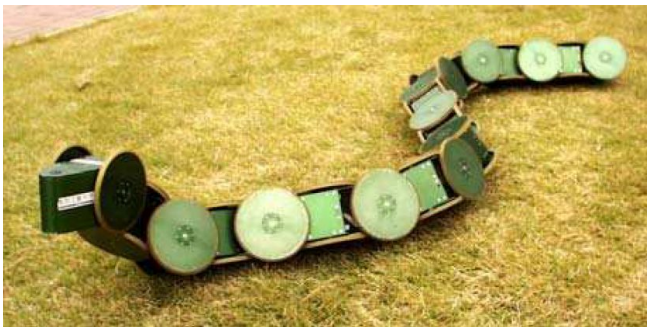


Figure 1.6: Hirose ACM3.

One of the possible configurations of an organism is a snake like robot. This configuration could be used to overcome obstacles, to move in an area with narrow passages or to move more swiftly. The robots could assemble in order to move through difficult terrain and dissasemble on the spot to fullfill different tasks.

1.4 Goals

We try to find a simple solution for adaptive snake-like locomotion. Making the equation for snake-like locomotion easier would make it easier to find a fitting fitness function. The challenges are to coordinate the degrees of freedom (DOF) with a central pattern generator (CPG) into a snake-like motion, to track down the target with a compass and to avoid obstacles. A first approach for the CPG was to use the periodic characteristics of the logistic map to induce the undulation of the snake. Then we fed the CPG with the common sinus generator.

Figure 1.7: Slim Slime robot.

In addition this should be a instruction for future students who want to work with the snake robot.

1.5 Organization of this thesis

This thesis consists of 5 Chapters:

- Chapter 1 gives an introduction into the problem setting and its environment, like former, present and future work and projects.
- Chapter 2 is about the theoretical foundations of the snake-like locomotion, obstacle avoidance and the first approach for oscillation induction
- Chapter 3 is about the implementations developed during this thesis. The implementations were done on the snake robot build by Meister and Stepanenko.
- Chapter 4 explains the experiments done with the snake robot and their results.
- Chapter 5, finally, contains a conclusion where the most important items are stressed and gives hints for further studies.

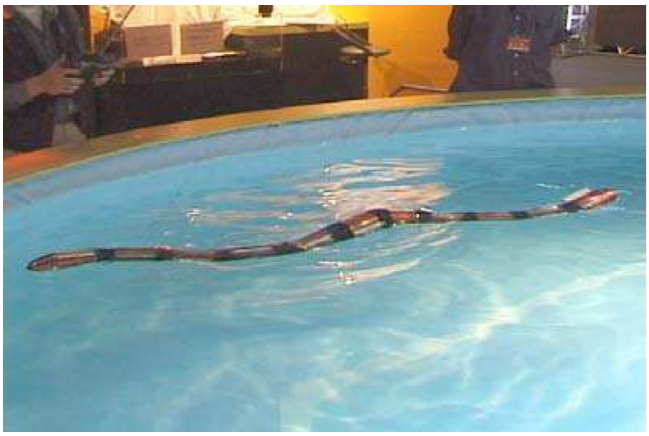


Figure 1.8: HELIX.

2 Foundations

In this chapter the theoretical foundations of snake-like locomotion are described as well as the basic theory of the hardware and software design of the robot. In detail we observe the movement of real snakes to bring it to our robot, in this case the CPG on the PSOC First Touch Starter Kit.

2.1 Central Pattern Generator

In neuroatonomy Central pattern generators (CPGs) are neural networks that produce rhythmic patterned outputs without sensory feedback. CPGs have been shown to produce rhythmic outputs resembling normal "rhythmic motor pattern production" even in isolation from motor and sensory feedback from limbs and other muscle targets. [Hooper, 2010] In robotics CPGs generate patterns for the locomotion or other complex behavior. As well as a human doesn't think about how he walks in detail, the robot also uses the pattern from the CPG to move. The several DOFs that are used for locomotion are controlled by the CPG. To achieve a certain goal you can influence the CPG by adjusting the parameters.

When constructing a CPG model, one has to define the following items: (1) The general architecture of the CPG. This includes the type and number of oscillators or neurons. In a robot, it also involves choosing between position control (i.e. the outputs of the CPG are the desired joint angles provided to a feedback controller) or torque control (i.e. the outputs of the CPG directly control the torque produced by the motors). (2) The type and topology of couplings. These will determine the conditions for synchronization between oscillators and the resulting gaits, i.e the stable phase relations between oscillators. (3) The waveforms. These will determine what trajectories will actually be performed by each joint angle during a cycle. The waveforms are clearly dependent on the shape of the limit cycle produced by the chosen (neural) oscillator, but can be transformed by the addition of filters. (4) The effect of input signals, i.e. how control parameters can modulate important quantities such as the frequency, amplitude, phase lags (for gait transition), or waveforms (e.g. for independently adjusting swing and stance phases). (5) The effect of feedback signals, i.e. how feedback from the body will affect the activity of the CPG (for instance accelerating or decelerating it depending on environmental conditions). A major difficulty in designing CPGs is that these five design axes are all strongly interconnected. [Ijspeert, 2008]

In our case the CPG controls the angle between the modules of the snake. All alignments to the topology of the robot (orientation, basic position of the motors) are stated either in

the design of the PSOC or in the code. As oscillator a sinus waveform was chosen and the pattern repeats after 360 steps. There are two parameters that can change while the robot is moving autonomous: (1) the basic moving direction, by default the 360 steps are increasing, if necessary the robot changes to decreasing steps to move backwards. (2) the bias Y for change of global direction for target tracking and obstacle avoidance.

2.2 Logistic map

A first approach in the research for this thesis was to find parameters for the logistic map. Which could induce an oscillation to the Central Pattern Generator in exchange for the sinus-function. We found several interesting patterns which showed periodic

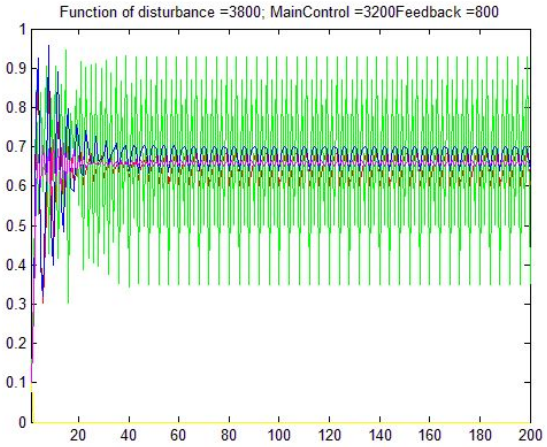


Figure 2.1: Logistic map on a snake with 5 modules

characteristics. But none of them was fitting for the locomotion of the snake. So after a few weeks of research we moved on to the common sinus-function.

2.2.1 Snake-like locomotion

The first snake like robot was built by Shigeo Hirose in december 1972. He found many of the basics of snake-like locomotion: The movement of a snake can be categorized into the following four types: 1) serpentine movement, 2) rectilinear movement, 3) concertina movement, and 4) side winding movement. Out of these, the serpentine movement is the most smooth and obviously effective mode of locomotion, and it is widely observed in almost all species of snakes. [Hirose and Yamada, 2009] In this thesis we concentrate on the serpentine movement. The central equation for serpentine movement is the following[Eugen Meister, 2011]:

Definition 2.1 (Snake motion)

The equation of serpentine can be written as follows:

$$x(t) = \int_0^t cos(\alpha)dt, y(t) = \int_0^t sin(\alpha)dt,$$

where $\alpha_i = \alpha_m \cdot sin(\omega t + \varphi(i-1)), i = 1, ..., N$
By adding a bias to the formula of calculation of the relative angle between the segments we can change the global direction of motion.

$$\alpha_i = \alpha_m \cdot sin(\omega t + \varphi(i-1)) + \Upsilon, i = 1, ..., N$$

In these equations:

- t - simulation time
- x, y - coordinates of the turning nods of the snake
- α - relative angels between segments of the snake
- α_m
- ω and φ affect the number of periods in one snake body
- N - number of segments in the snake body
- Υ - Bias for global direction change

2.3 Target tracking

The robot has only a compass for orientation, so we let him snake into one direction. This was realized by following equation.

Definition 2.2 (Bias computing target tracking)

The bias which has direct influence on the change of direction of the snake can be computed as follows:

$$\Upsilon = \begin{cases} \Upsilon_{Max}, & \textbf{if } (\tau_{Target} - \tau) > 90 \\ -\Upsilon_{Max}, & \textbf{if } (\tau_{Target} - \tau) < -90 \\ \frac{\Upsilon_{Max} \cdot (\tau_{Target} - \tau)}{90}, & \textbf{if } -90 \leq (\tau_{Target} - \tau) \leq 90 \end{cases}$$

In these equations:

- Y - Bias
- τ - current heading of the snake
- τ_{Target} - direction towards the target
- Y_{Max} - Maximum bias the snake can take

A target tracking inside the arena would be possible by implicate the speed of the robot, but due to the sensitivity of the robot to little changes of the slip on the ground it is impossible to compute. For further research a light sensor could be added to the head of the robot, so he could search for the brightest spot in the arena.

2.4 Obstacle avoidance

In this thesis we use a simple algorihm for obstacle avoidance which was also used by Eugen Meister for his article "Adaptive Locomotion of Multibody Snake-like Robot". It's based on the exponential function so the reaction to an obstacle would increase swiftly when the robot approaches. As soon as an obstacle is detected the bias is only changed by the obstacle avoidance until the robot has passed by.

Definition 2.3 (Bias computing obstacle detection)
The bias which has direct influence on the change of direction of the snake can be computed as follows:

$$Y = sgn(\tau_{Head}) \cdot Y_{Max} \cdot e^{\left(\frac{a}{\rho}\right)}$$

In these equations:

- Y - Bias
- τ_{Head} - current heading of the snake head
- Y_{Max} - Maximum bias the snake can take
- ρ - Distance between snake head and obstacle
- a - scale factor for the sensitivity of the snake

If the robot faces an obstacle less than 5 cm away, the snake goes revers. This shouldn't be necessary if the parameters are set well.

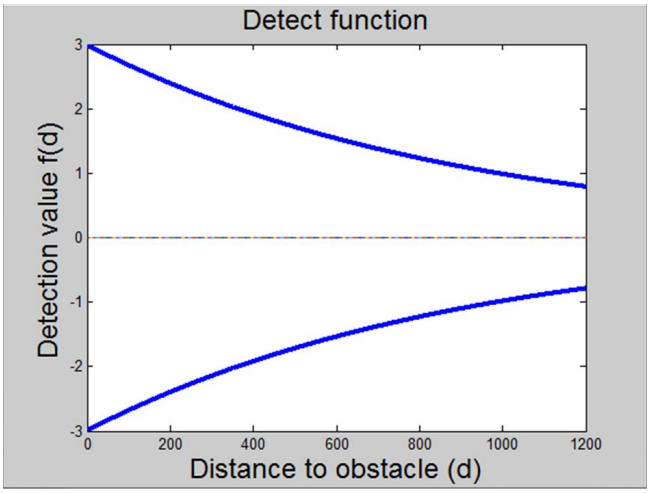


Figure 2.2: Detect function

3 Implementation

In this chapter we describe the implementations on the snake robot. Based on the work of Sergej Stepanenko and Eugen Meister we changed and improved the programming of the snake robot. Following implementations have been written and tested:

- Communication between PSOC and Compass via I^2C.
- Control-output for test via UART
- Different approaches for the compass to bring meaningful data
- Central Pattern Generator
- Obstacle avoidance

3.1 PSOC

As the hardware base the new development of company Cypress Semiconductor was used. This is the so-called Programmable System-on-Chip (PSoC). PSoC is customizable analog and digital periphery, memory and microcontroller on the one chip. The advantage of this system is fully programmable hardware part, there are no restrictions on the use of the periphery. PSoC has the following advantages:

- Routing and Interconnect - this frees to re-route signals to userselected pins, shedding the constraints of a fixed-peripheral controller. In addition, global buses allow for signal multiplexing and logic operations, eliminating the need for a complicated digital-logic gate design.
- Configurable Analog and Digital Blocks. The union of configurable analog and digital circuitry is the basis of the PSoC platform. You configure these blocks using pre-built library functions or by creating your own. By combining several digital blocks, you can create 16-, 24-, or even 32-bit wide logic resources. The analog blocks are composed of an assortment of switch capacitor, op-amp, comparator, ADC, DAC, and digital filter blocks, allowing complex analog signal flows. In PSoC software includes a great number of preset blocks, that can be changed and adapted to the needs of exactly the scheme that is developed.
- CPU Subsystem. PSoC offers a sophisticated CPU subsystem with SRAM , EE PROM, and flash memory, multiple core options and a variety of essential system resources including:

  - Internal main and low-speed oscillator;
  - Connectivity to external crystal oscillator for precision, programmable clocking;
  - Sleep and watchdog timers;
  - Multiple clock sources that include a PLL.

- PSoC devices also have dedicated communication interfaces like I2C, Full-Speed USB 2.0, CAN 2.0, and on-chip debugging capabilities using JTAG and Serial Wire Debug. The newest members of the PSoC family offer industry-standard processors like the 8051 and ARM Cortex-M3.

[Eugen Meister, 2011]

3.2 I²C - Communication

I²C ("i-squared cee" or "i-two cee"; Inter-Integrated Circuit; generically referred to as "two-wire interface") is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, cellphone, or other electronic device. [WIKIPEDIA, 2008] I²C defines three basic types of messages, each of which begins with a START and ends with a STOP:

- Single message where a master writes data to a slave;
- Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves.

For the first type of message, that was frequently used for communication with the compass, we wrote the following procedure:

```
void I2C_Send(uint8 adr, uint8 first, uint8 second)
{
  uint8 status;
  uint8 i;
  I2C_1_Start();
  I2C_1_MasterClearStatus();

  ////
  Resister1[0]=first; //move pointer to the CRA register
  Resister1[1]=second; //CRA6-5 averaged Samples, CRA4-2 Output Rate, CRA1-0
      Measurement Mode

  CYGlobalIntEnable;
  ////////////////////////////////////////////////////////
  I2C_1_MasterClearStatus();
  status = I2C_1_MasterSendStart(adr, I2C_1_WRITE_XFER_MODE);
```

```
  if(status == I2C_1_MSTR_NO_ERROR) // Check if transfer completed without
      errors
  {
  // Send data for configuration
    for(i=0; i<2; i++)
    {
    status = I2C_1_MasterWriteByte(Resister1[i]);
      if(status != I2C_1_MSTR_NO_ERROR )
      {
        break;
      }
    }
  }
  I2C_1_MasterSendStop(); // Send Stop
return;
}
```

We followed the protocol described in the datasheet of the HMC5883L - compass. The procedure sends a write request and after the acknowledgement it sends two byte data. The first one is the address of the register we want to write to, the second the data that should be written. The communication period with the compass can be seen in the following graphic.

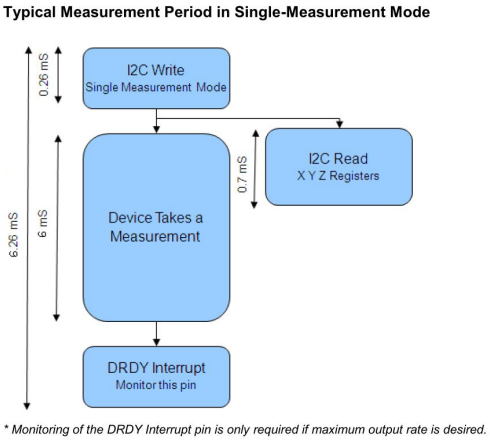


Figure 3.1: I2C Communication Period

3.3 UART

A universal asynchronous receiver/transmitter, abbreviated UART, is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. [WIKIPEDIA, 2008] It's a simple to use communication standard which is already included on the PSOC. It can be used as Input and Output, but as we are using it only for observation we only implemented an Output. Disregarding

the standard UART functions from the library. We wrote a little procedure to display double variables.

```
UART_PrintD(double Number)
{
  if (Number <0){ UART_1_PutChar(45);Number =-Number; }

    uint8 h = (uint8) (Number/100);
    uint8 d = (uint8) (Number/10 - 10*h);
    uint8 e = (uint8) (Number-(d*10+h*100));
    uint8 z = (uint8) (Number*10-(e*10+d*100+h*1000));
    UART_1_PutChar(h+48);
    UART_1_PutChar(d+48);
    UART_1_PutChar(e+48);
    UART_1_PutChar(44);
    UART_1_PutChar(z+48);
}
```

3.4 Initialisation

For computing of many standard functions like squareroot or arctangent we had to include the math.h - library. The device.h and stdio.h are necessary for every programm on PSOC for communication between analog and digital blocks and processing unit.

```
#include <math.h>
#include <device.h>
#include <stdio.h>
```

There are many static values such as addresses and maximum values that weren' t changed during action.

```
#define Period_PWM 40000
//#define MAX_Amp 1100
#define MAX_Step 360
//#define Compass_Addr 0x42
#define Front_Sonar 0x70
#define Back_Sonar 0x71
#define Front_Sensor 0
#define Back_Sensor 13
#define Down_1_Sensor 14
#define Down_2_Sensor 15
#define MAX_Detect 220
#define Critical_Detect 120 // 120 old value
#define DETECT 60 //60 old value
#define MAX_IgnoreSensor 2
```

Init starts the communication protocols, configurates the compass for permanent measurement and sets the snake to a straight shape.

```
void Init(void)
{
```

```
I2C_1_EnableInt();
I2C_1_Start();
UART_1_Start();
ISR_IR_Start();

IR_Int=255;
CYGlobalIntEnable;



PWM_6_Start();
CyDelay(Init_Delay);
PWM_7_Start();
CyDelay(Init_Delay);
PWM_5_Start();
CyDelay(Init_Delay);
PWM_8_Start();
CyDelay(Init_Delay);
PWM_4_Start();
CyDelay(Init_Delay);
PWM_9_Start();
CyDelay(Init_Delay);
PWM_3_Start();
CyDelay(Init_Delay);
PWM_10_Start();
CyDelay(Init_Delay);
PWM_2_Start();
CyDelay(Init_Delay);
PWM_11_Start();
CyDelay(Init_Delay);
PWM_1_Start();
CyDelay(Init_Delay);
PWM_12_Start();
CyDelay(Init_Delay);
PWM_13_Start();
CyDelay(Init_Delay);
Move_All();
CyDelay(Init_Delay);

Counter_1_Start();
Counter_2_Start();

AMux_Start();
AMux_Select(1);
ADC_Start();
ADC_StartConvert();
CyDelay(100);

dStep=0;
//////////////////////////////////////////
// Config Compass

I2C_Send(0x1E, 0x00, 0x70); //CRA6-5 averaged Samples, CRA4-2 Output Rate,
    CRA1-0 Measurement Mode
```

```
  I2C_Send(0x1E, 0x01, 0xA0);//CRB7-5 gain

  I2C_Send(0x1E, 0x02, 0x00);//0x01..single converstation,0x00..conti-mode

  return;
}
```

All infrared sensors are connected to a AMUX-block. To read the data Get_Sensor calls the data from the block and returns it.

```
uint8 Get_Sensor(uint8 Addr)
{

  int16 Buf;
  AMux_Select(Addr);
  ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
  Buf=ADC_GetResult16();
  if (Buf<0)
    return 0;
  return (uint8) Buf;
}
```

3.5 IR-Remote

The IR-Remote is used to initialise the movement of the snake and can be used to remotely change bias and amplitude.

```
    ////////////////////////////////////
    // IR-Remote
    ////////////////////////////////////
    if (IR_Int!=255)
    {
      ISR_IR_Disable();
      switch (IR_Int)
      {
        case 132: // Cancel
             CyDelay(100);
             break;
        case 204: // V -> S
             CyDelay(100);
             break;
        case 236: // Set
             CyDelay(100);
             break;
        case 156: // M1
             Step++;
             break;
        case 220: // M2
             break;
        case 188: // M3
             break;
```

```
        case 252: // M4
             break;
        case 180: // Video
             Direction -= 0.3;
             break;
        case 148: // Audio
             Direction = 0;
             break;
        case 244: // Audio Mode
             Direction += 0.3;
             break;
        default: LED_Reg_Write(255);
      }
      ISR_IR_Enable();
      IR_Int=255;
    }
```

All commands are listed in the following table

3.6 Central Pattern Generator

The Central Pattern Generator sets the steps from 0 to 359.

```
    Move_All(); //Set ankle of PWM

    ////////////////////////////////////////
    // Reseting Step
    if ((dStep==-1)&(Step==0))
      Step = MAX_Step;
    Step+=dStep;
    if (Step>=MAX_Step)
      Step = 0;
}
```

3.6.1 Undulation equation

Move_All() computes the angle of every motor and ...

```
void Move_All(void)
{
  uint8 i;
  for (i=1; i<25; ++i)
  {
       Set_PWM(i, MAX_Amp*Amp[i]*(sin(pi/180*(Omega[i]*Step+(int)(i-1)/2*teta)*
           Omeg))+Direction);

    CyDelayUs(Delay);


  }
```

```
  return;
}
```

... Set_PWM allocates the adjusted data of the angles to the motors via Pulse-width modulation blocks.

```
void Set_PWM(uint8 Num, float Val)
{
  float Value=Val;
  switch (Num)
  {
    case 1: PWM_1_WriteCompare1(Period_PWM - PWM_def[1] - Value); break;
    case 2: PWM_1_WriteCompare2(Period_PWM - PWM_def[2] - Value); break;
    case 3: PWM_2_WriteCompare1(Period_PWM - PWM_def[3] - Value); break;
    case 4: PWM_2_WriteCompare2(Period_PWM - PWM_def[4] - Value); break;
    case 5: PWM_3_WriteCompare1(Period_PWM - PWM_def[5] - Value); break;
    case 6: PWM_3_WriteCompare2(Period_PWM - PWM_def[6] - Value); break;
    case 7: PWM_4_WriteCompare1(Period_PWM - PWM_def[7] + Value*1.1); break;
    case 8: PWM_4_WriteCompare2(Period_PWM - PWM_def[8] - Value); break;
    case 9: PWM_5_WriteCompare1(Period_PWM - PWM_def[9] - Value); break;
    case 10:PWM_5_WriteCompare2(Period_PWM - PWM_def[10] - Value); break;
    case 11:PWM_6_WriteCompare1(Period_PWM - PWM_def[11] + Value*1.1); break;
    case 12:PWM_6_WriteCompare2(Period_PWM - PWM_def[12] - Value); break;
    case 13:PWM_7_WriteCompare1(Period_PWM - PWM_def[13] + Value*1.1); break;
    case 14:PWM_7_WriteCompare2(Period_PWM - PWM_def[14] - Value); break;
    case 15:PWM_8_WriteCompare1(Period_PWM - PWM_def[15] - Value); break;
    case 16:PWM_8_WriteCompare2(Period_PWM - PWM_def[16] - Value); break;
    case 17:PWM_9_WriteCompare1(Period_PWM - PWM_def[17] + Value*1.1); break;
        //neuer Motor
    case 18:PWM_9_WriteCompare2(Period_PWM - PWM_def[18] - Value); break;
    case 19:PWM_10_WriteCompare1(Period_PWM - PWM_def[19] - Value); break;
    case 20:PWM_10_WriteCompare2(Period_PWM - PWM_def[20] - Value); break;
    case 21:PWM_11_WriteCompare(Period_PWM - PWM_def[21] - Value); break;
    case 22:PWM_14_WriteCompare(Period_PWM - PWM_def[22] - Value); break;
    case 23:PWM_12_WriteCompare(Period_PWM - PWM_def[23] - Value); break;
    case 24:PWM_13_WriteCompare(Period_PWM - PWM_def[24] - Value);
  }
  //CyDelay(1);
  return;
}
```

3.6.2 Obstacle avoidance

The obstacle avoidance lets the snake robot walk backwards if he is less than 5cm before an obstacle.

```
//////////////////////////////////////////
// Obstacle avoidance
CurrentSensorValue=Get_Sensor(DefSensor);

// the ranges are DETECT=30cm Critical_Detect=8cm MAX_Detect=2cm

// standing directly before obstacle
```

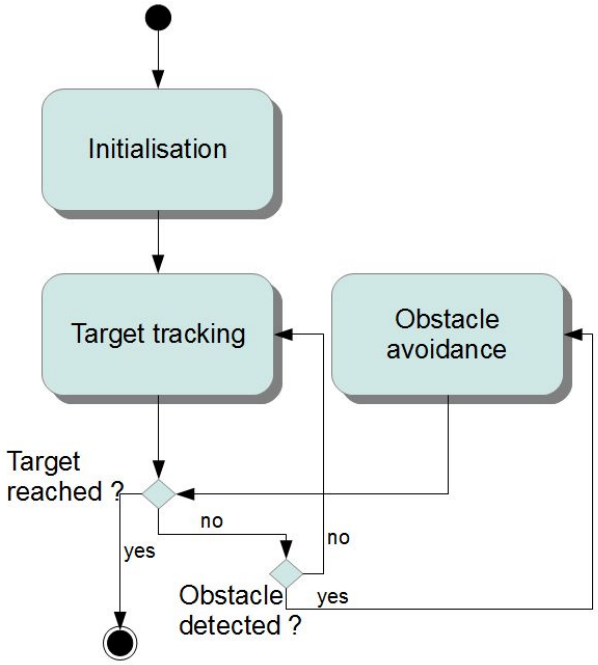


Figure 3.2: Locomotion figure..

```
if (CurrentSensorValue>MAX_Detect)
{
  //Step = 0;
  //dStep=0;//-dStep; // Stop moving if obstacle is directly in front of the
      snake
  if (dStep ==1)
  {
    CyDelay(100);
    dStep = -1; //walk reverse if obstacle is directly in front of the snake
    DefSensor = 13;
  }
  else
  {
    CyDelay(100);
    dStep = 1;
    DefSensor = 0;
  }

}
else
{
  // first Contact
  if (CurrentSensorValue>DETECT)
  {
    if ((Step < 45) | (Step > 225))
    {Direction = exp((CurrentSensorValue-DETECT)/300)*200;  }
```

```
    else
    {Direction = exp((CurrentSensorValue-DETECT)/300)*200; }

  }
  else
  {
    obj = target_obj;
    ////////////////////////////////////////////////
    // Targeting
    double deltaDir = (obj-heading);
    if (deltaDir > 180) deltaDir -= 360;
    if (deltaDir < -180) deltaDir += 360;

    if (deltaDir > 90) Direction = +0.8;
    else
    { if (deltaDir < -90) Direction = -0.8;
      else Direction = (float) deltaDir * 0.8/ 90; //deltaDir*0.5/90
    }
  }
}
```

3.7 Compass

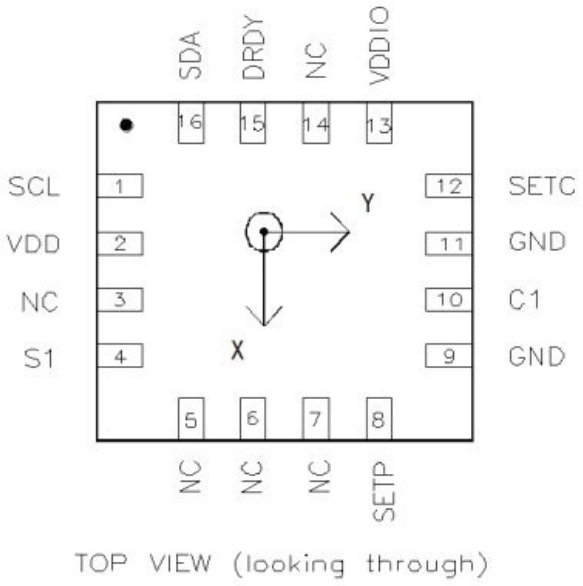


Figure 3.3: HMC5883L.

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as compassing and magnetometry. The HMC5883L includes state-of-the-art, high-resolution HMC118X series

magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1 to 2 degree compass heading accuracy. The I^2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.
The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.
The chip gives 6 I^2C packages (each 1 byte) which were put together to 3 measurements of the 3 magnetometers arranged in the 3 axes with values from -2048 to 2047. The compass gives a 3-dimensional vektor which should point to the magnetic north pole. **Features:**

- 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package
- 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in +/-8 Gauss Fields
- Built-In Self Test
- Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 $\hat{I}\frac{1}{4}A$)
- Built-In Strap Drive Circuits
- I^2C Digital Interface
- Lead Free Package Construction
- Wide Magnetic Field Range (+/-8 Oe)
- Software and Algorithm Support Available
- Fast 160 Hz Maximum Output Rate

Benefits:

- Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT CapacitorsDesigned for High Volume, Cost Sensitive OEM Designs Easy to Assemble and Compatible with High Speed SMT Assembly
- Enables 1 to 2 Degree Compass Heading Accuracy
- Enables Low-Cost Functionality Test after Assembly in Production

- Compatible for Battery Powered Applications
- Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
- Popular Two-Wire Serial Data Interface for Consumer Electronics
- RoHS Compliance
- Sensors Can Be Used in Strong Magnetic Field Environments with a 1 to 2 Degree Compass Heading Accuracy
- Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
- Enables Pedestrian Navigation and LBS Applications

3.7.1 Placement

The compass was placed on the head of the snake so we could get the current viewing direction. The X points to the front the Y to the right. Due to the magnetic disturbances of the electic motors and wire harness of the snake robot and other disturbances in the environment, we placed a copper shield below the compass. The shield should restrain the magnetic fields from the robot that influenced the compass.

3.7.2 Connection

The compass is controlled via the I^2C bus. This device will be connected to this bus as a slave device. The address is 0x3C but only the first seven bit are used in the I^2C-Protokoll, so for the communication from PSOC to compass we use the address 0x1E (0x3C devided by 2).
The compass is intialised in Init()-Function for permanent use. He provides the data until it is read. Not till then he takes the next measurement.

```
uint8* Compas(uint8 adr)
{
   uint8 Resister1[2];
  uint8 status;
  uint16 delay;
  uint8 i;
  UART_1_Start();

  ////////////////////////////////////////////////////////////
  // Read back the data
  I2C_1_MasterClearStatus();
  //CyDelayUs(10000);
  status = I2C_1_MasterSendStart(adr, I2C_1_READ_XFER_MODE);
  if(status == I2C_1_MSTR_NO_ERROR) // Check if transfer completed without
      errors
```

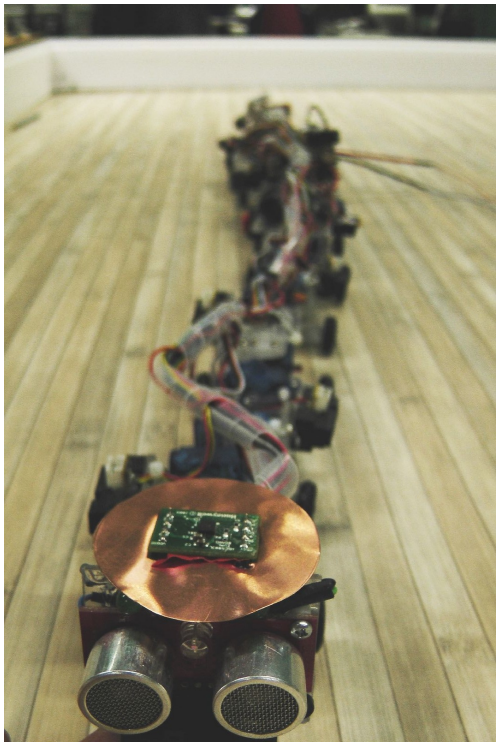


Figure 3.4: Snake with compass

```
  {
    for(i=0; i<6; i++)
    {
      if(i < 6)
      {
      //CyDelayUs(10000);
        CompasValue[i] = I2C_1_MasterReadByte(I2C_1_ACK_DATA);
      }
      else
      {
        CompasValue[i] = I2C_1_MasterReadByte(I2C_1_NAK_DATA);
        UART_1_WriteTxData(CompasValue[i]);
      }
    }
  }

  I2C_1_MasterSendStop();

  status = I2C_1_MasterSendStart(adr, I2C_1_WRITE_XFER_MODE);
  status = I2C_1_MasterWriteByte(0x03);
  I2C_1_MasterSendStop();
  return CompasValue;
}
```

This procedure sends a request to read to the compass. If all is clear, it reads all 6 registers, sets the pointer back to the first register and returns the raw data.

```
double Get_Compass(void)
{

  UART_1_Start();

  CompasVal = Compas(0x1E);
  int16 realX, realY, realZ;

  realX = (CompasVal[0] << 8) | CompasVal[1];
  realZ = (CompasVal[2] << 8) | CompasVal[3];
  realY = (CompasVal[4] << 8) | CompasVal[5];
  uint8 i;

  double heading;
  double rX = (double) realX;
  double rY = (double) realY;
  double rZ = (double) realZ; // uncomment this to compute gradient

  heading = atan2(rY,rX) * 180 / pi;
  double grad = 90 - (atan (rZ/(sqrt (rX*rX + rY*rY) ))* 180 / pi); //
      uncomment this to compute gradient

  if (heading < 0) heading += 360;

  return heading;
}
```

This procedure puts the raw data together and computes the heading using the atan2-function. The atan2-function is a variation of the arctangent-function to compute the angle of the gradient out of the x- and y-values of the magnetometer.
To check the HMC5883L for proper operation, a self test feature is incorporated in which the sensor offset straps are excited to create a nominal field strength (bias field) to be measured. To implement self test, the least significant bits (MS1 and MS0) of configuration register A are changed from 00 to 01 (positive bias) or 10 (negative bias).

```
void Compass_Self_Test(void)
{
  uint8 i;

  I2C_Send(0x1E, 0x00, 0x71);
  I2C_Send(0x1E, 0x01, 0xA0);
  I2C_Send(0x1E, 0x02, 0x00);
  for (i=0;i<100;i++)
  {
  CompasVal = Compas(0x1E);
  int16 realX, realY, realZ;

  realX = (CompasVal[0] << 8) | CompasVal[1];
  realZ = (CompasVal[2] << 8) | CompasVal[3];
  realY = (CompasVal[4] << 8) | CompasVal[5];
```

```
  uint8 i;

  UART_1_WriteTxData(255);
  for (i=0;i<6;i++)
  UART_1_WriteTxData(CompasVal[i]);

  double heading;
  double rX = (double) realX;
  double rY = (double) realY;
  double rZ = (double) realZ; // uncomment this to compute gradient

  UART_1_PutString("X:␣");
  UART_PrintD(rX);
  UART_1_PutChar(0x09);
  UART_1_PutString("Y:␣");
  UART_PrintD(rY);
  UART_1_PutChar(0x09);
  UART_1_PutString("Z:␣");
  UART_PrintD(rZ);
  UART_1_PutCRLF(32);
  }
}
```

The default gain is 1. If the output values are outside of the range (specified by the following chart) you must increase the gain.

3.7.3 Error correction

To try the compass we mostly turned the device in 90 degrees and compared the output. At the first try on the testboard the compass worked quite well. As the image shows the angles are nearly perpendicular.

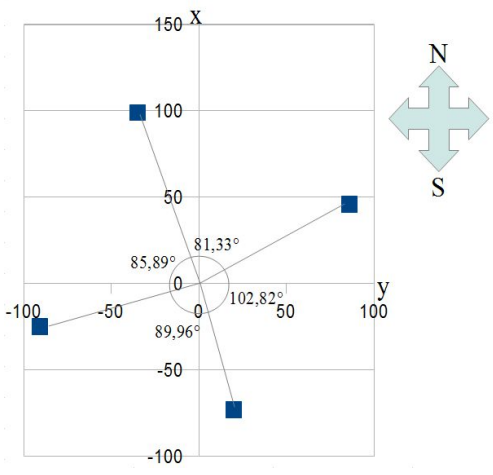


Figure 3.5: Compass data on testboard

After we connected and calibrated the compass we tried it again with motors online and

offline and the disturbances were high. After several test we realized that the magnetic field of the snake robot adds an offset to the data.

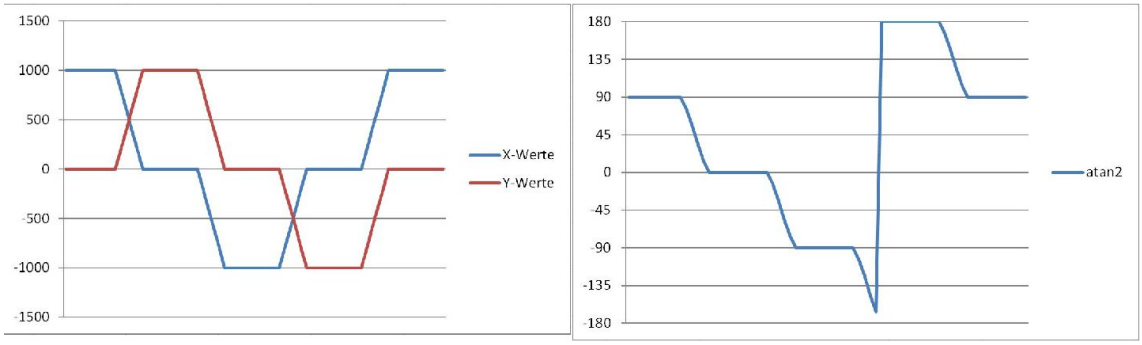


Figure 3.6: Compass data expected

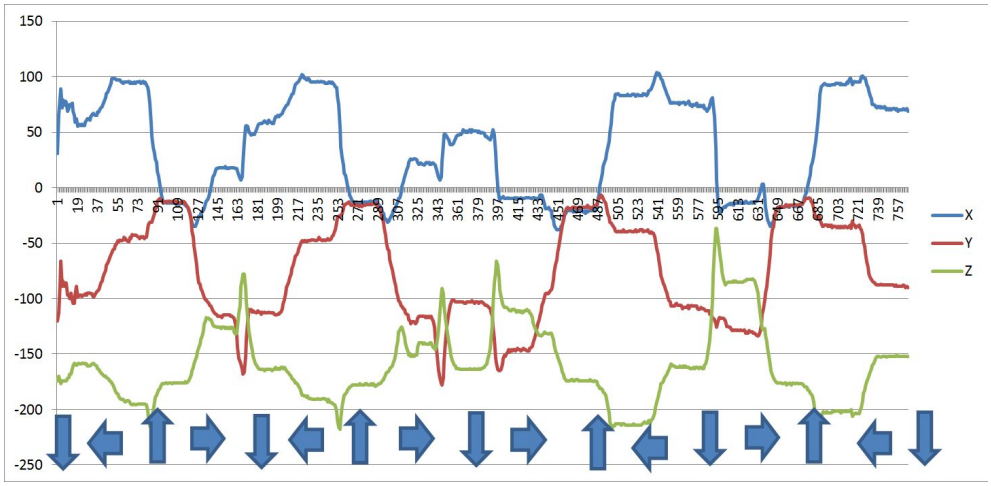


Figure 3.7: Compass real data

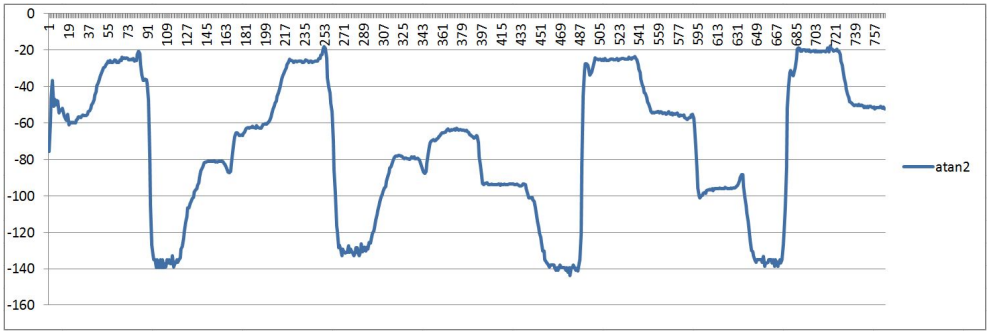


Figure 3.8: Compass angle from real data

Definition 3.1 (Compass offset computing)

The offset can be computed. You must make a full rotation with the robot and the offset is the arithmetic average of the maximum and minimum:

$$x_{Offset} = \frac{x_{Min}+x_{Max}}{2}$$

$$y_{Offset} = \frac{y_{Min}+y_{Max}}{2}$$

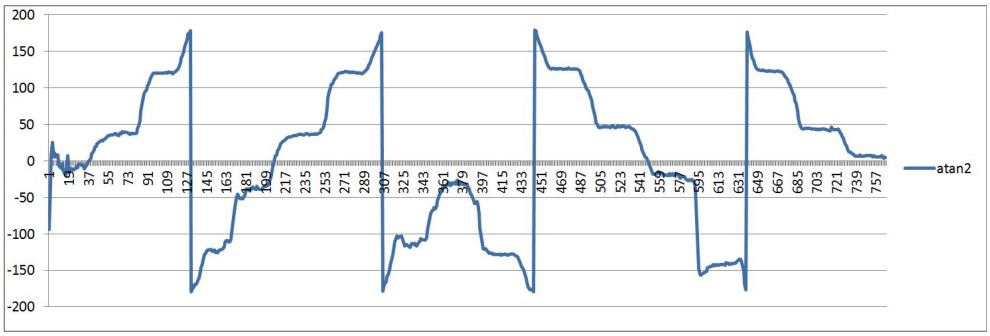


Figure 3.9: Compass angle with offset compensation

By this correction we got usable measurements for the robot. But when we tried to let the snake walk in one direction, the measurement became again unreadable. Most likely because of the greater magnetic influence of the motors when they change their angle and the changing configuration of the snake robot when he bends into a direction (influence of the bias).
The next approach was to add the vectors and get a smoothend reading from several measurements. Unfortunately the measured vectors werent normalised (had different length) so the resulting vector wasn't meaningful.
Finally we buffered 36 measurements and took the average of the heading of one period. By this we could achieve a snake robot who can walk into any given direction.

```
////////////////////////////////////
// Compass Processing
double buffer;
if ((Step%10) == 0)
{
  buffer = Get_Compass(1);
  heading_buffer[(int)(Step/30)]=buffer;
  double head = 0;
  for( i=0; i<36; i++)
  {
    head += heading_buffer [i];
  }
```

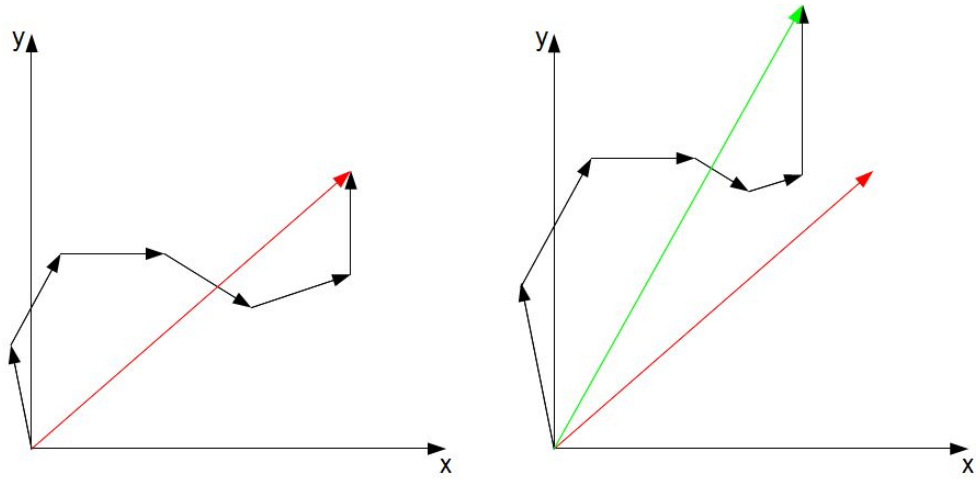


Figure 3.10: Resulting vector as wished and real

```
  heading = head /36;
}
```

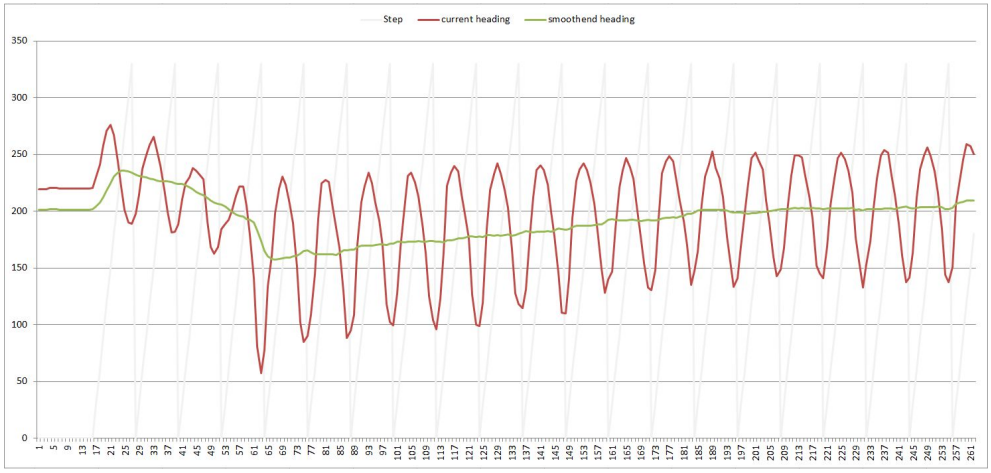


Figure 3.11: Resulting vector as wished and real

Button	Code	Command	Effect
Cancel	132	CyDelay(100);	Delay
V -> S	204	CyDelay(100);	Delay
Set	236	CyDelay(100);	Delay
M1	156	Step++;	Manual moving
M2	220	-	-
M3	188	-	-
M4	252	-	-
Video	180	Direction -= 0.3;	Turn left
Audio	148	Direction = 0;	Walk straight
Audio Mode	244	Direction += 0.3;	Turn right
Power	212	Stop_Moving;	Stop
Clear	248	dStep=1; Start_Moving;	Moving forward
Front/Rear	172	dStep=-1; Start_Moving;	Moving backward
Mode Lock	140	-	-
Matrix	235	-	-
Audio Att down	164	PWM1_Val-=50;	Turn head left
Audio Att up	228	PWM1_Val+=50;	Turn head right
In/Out 2	149	-	-
1	128	Max_Amp=600;	Amplitude change
2	192	Max_Amp=700;	Amplitude change
3	160	Max_Amp=800;	Amplitude change
4	224	Max_Amp=900;	Amplitude change
5	144	Max_Amp=1000;	Amplitude change
6	208	Max_Amp=1100;	Amplitude change
Monitor1 1	129	Omeg=1;	Omega change
Monitor1 2	193	Omeg=2;	Omega change
Monitor1 3	161	Omeg=4;	Omega change
Monitor1 4	225	Omeg=6;	Omega change
Monitor1 5	145	Omeg=8;	Omega change
Monitor1 6	209	Omeg=10;	Omega change
Monitor2 1	137	Omeg=12;	Omega change
Monitor2 2	201	Omeg=14;	Omega change
Monitor2 3	169	Omeg=16;	Omega change
Monitor2 4	233	Omeg=18;	Omega change
Monitor2 5	153	Omeg=20;	Omega change
Monitor2 6	217	Omeg=22;	Omega change
Processor	243	Delay=50;	Delay change (fast)
Pass Video	139	Delay=100;	Delay change
Pass Audio	203	Delay=200;	Delay change
Mode-C V/A	221	Delay=400;	Delay change (slow)

Table 3.1: Range of Limits after Self Test

Gain	Low limit	High Limit
0	854	2020
1	679	1607
2	511	1209
3	411	973
4	274	648
5	243	575
6	206	487
7	143	339

Table 3.2: Range of Limits after Self Test

4 Experiments

In this chapter we describe the experiments made with the robots. Some of them are already partially described in the previous chapter

4.1 Arena

The Arena is 2m x 4m. The ground is made of laminate. She is sourounded by 13cm high walls. The laminate gives good grip for the serpentine movement of the snake robot and compensates little bruises from the wooden shelves under the ground.

4.2 Walking straight

After changing several parts of the robot the first goal was to adjust the new parts so the robot would walk straight. At this exercise the robot showed to be very sensitive to down-grade. The arena was slightly sloping and so the robot turned downhill.

4.3 Target tracking

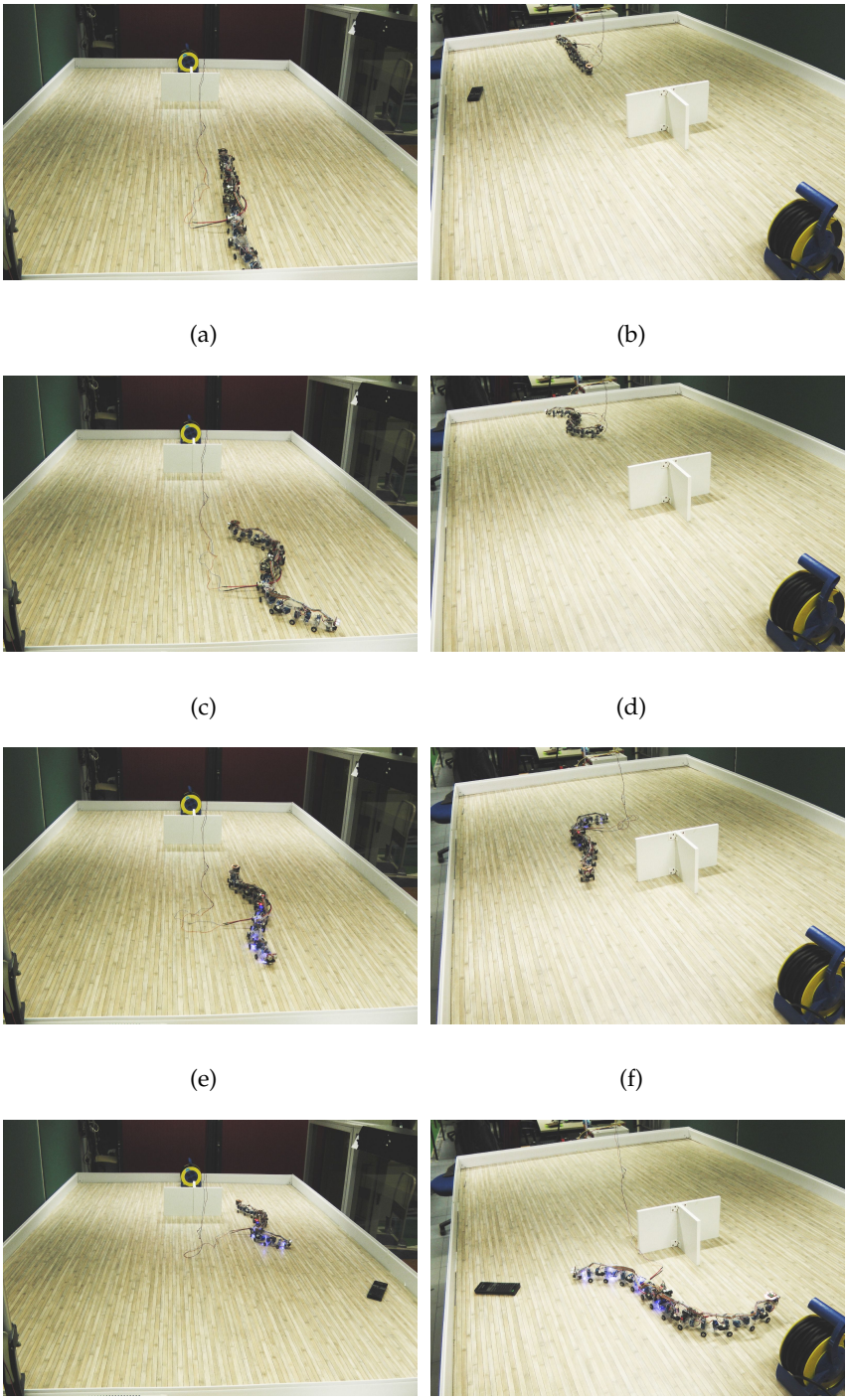
As described in the previous chapter there were several difficulties adjusting the compass.
The first setting was only the compass connected to another PSOC testboard. The compass gave back comprehensable measurement data, but not within the expected 1 to 2 degrees reliability but 5 to 10 degrees difference. He also showed significant sensitivity when tipped out of the z-plane.
The second setting was on the snake with motors shut down. Anyway the wire harness and the motors induced even offline high distubances. We anticipated that the noise would be even greater with motors online and went straight to the next setting. The third setting was the snake with online motors in serpentine shape, head facing forwards (step=45). The snake stood on a board with a central joint to the ground as center of rotation. This made it possible to circle the snake robot around without much traction. After several experiments we realized out of the data, that we must add an offset to the magnetometer data for compensation.

4.4 Target tracking and obstacle avoidance

In the fourth setting, after computing and compensating the offset, we took measurements from the moving snake robot and again there were high disturbances. Most likely due to the moving motors and the bias changing the shape of the snake.
With the buffering method described in the previous chapter we finally got usable data from the compass and the snake started moving arround obstacles. Finally we were able to adjust the parameters from the obstacle avoidance. And the snake turned away from the obstacle and after passing it returned to the previous direction.

4.5 Results

The current drain is very high for such a small robot. The robot draw over 2A at a voltage of 12V.
The implemented sinus-function for the Central Pattern Generator leads to a robust movement. It is vulnerable for down-grade but the target tracking can compensate it.
Although the function for adaptive snake-like locomotion is very adaptable, the adaption to another robot is still difficult.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 4.1: (a),(c),(e),(g): Obstacle avoidance from front perspective,(b),(d),(f),(h): Obstacle avoidance from back perspective.

5 Conclusion

This work is an other different approach for adaptive snake-like locomotion. We hope that it leads for a better understanding of the adaptive snake-like locomotion. And contributes to the research in the field of multi-robot organisms.

5.1 Summary

The work of Sergej Stepanenko was enhanced and the experience with the compass adaption will hopefully help future students to get a better understaning and help them to adjust them swiftly to their projects.

5.2 Further research

Snakes all around Earth show how sucessful this lifeform is. Not at least cause of their locomotion. With regard to the research swarm robots and multi-robot organisms with evolutionary behavior there's still need to simplify the patterns for fitting fitness functions.
The serpentine movement is not the only locomotion snakes are capeable of. There's also the rectilinear movement, the concertina movement, and the side winding movement. If there's a next generation of the snake robot the motors and the frame will hopefully be much stronger so that the snake can move with all four movements.

5.2.1 Expandability

The existant snake robot could be ubgraded with a light sensor. The robot could find the brightest spot in the arena.
For future applications it would be nice if the robot could determine his absolute position.

Bibliography

[Ants, 2008] Ants (2008). http://en.wikipedia.org/wiki/image:leafcutter_ants_transporting_leaves.jpg. *(October 13, 2008)*.

[ATMEL, 2008] ATMEL (2008). http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf. *(July 29, 2008)*.

[Bees, 2008] Bees (2008). http://upload.wikimedia.org/wikipedia/commons/5/51/bees.jpg. *(August 29, 2008)*.

[Bell, 1997] Bell, G. (1997). *Selection: The Mechanism of Evolution*. Chapman and Hall.

[Beurer, 2008] Beurer, D. (2008). Development of hardware and software protocols for ZigBee communication in multi-robot organisms. *Diplomarbeit*, 2745.

[Birds, 2008] Birds (2008). http://lh3.ggpht.com/_0dxj-zo8sqc/r-grmnv4x0i/aaaaaaaacga/dotlaalw49i/dscf6930.jpg. *(August 29, 2008)*.

[CODESOURCERY, 2008] CODESOURCERY (2008). http://www.codesourcery.com/gnu_toolchains/arm/portal/subscription@template=lite. *(July 29, 2008)*.

[Cogger, 1991] Cogger, H. (1991). *Reptiles and Amphibians*, page 175. Smithmark Pub.

[Corne and Fogel, 2002] Corne, D. W. and Fogel, G. B. (2002). *An Introduction to Bioinformatics for Computer Scientists*, pages 3–18. Morgan Kaufman.

[CORTEX, 2008] CORTEX (2008). http://www.luminarymicro.com/home/data_sheets.html. *(July 29, 2008)*.

[Dawkins, 1987] Dawkins, R. (1987). *Der blinde Uhrmacher*. Kindler Verlag GmbH, Mï¿½nchen.

[E-PUCK, 2008] E-PUCK (2008). http://www.e-puck.org/. *(October 12, 2008)*.

[Eugen Meister, 2011] Eugen Meister, Sergej Stepanenko, S. K. (2011). Adaptive locomotion of multibody snake-like robot. *IEEE Press*.

[Fish, 2008] Fish (2008). http://notexactlyrocketscience.files.wordpress.com/2006/10/nwhi_-_french_frigate_shoals_reef_-_many_fish.jpg. *(August 29, 2008)*.

[Fogel and Corne, 2002] Fogel, G. B. and Corne, D. W. (2002). *An Introduction to Evolutionary Computation for Biologists*, pages 19–38. Morgan Kaufman.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, Inc.

[FreeRTOS, 2008] FreeRTOS (2008). http://www.freertos.org/. *(October 15, 2008)*.

[Hauser, 1996] Hauser, M. D. (1996). *The Evolution of Communication*. The MIT Press.

[Heschl, 1998] Heschl, A. (1998). *Das intelligente Genom*. Springer-Verlag Berlin Heidelberg.

[Hirose and Yamada, 2009] Hirose, S. and Yamada, H. (2009). Snake-like robots: A tutorial. *IEEE Robotics and Automation Magazine*, March 2009.

[Holland, 1975] Holland, J. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[Hooper, 2010] Hooper, S. (1999-2010). *Encyclopedia of Life Sciences*. John Wiley and Sons.

[Ijspeert, 2008] Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: A review. *Elsevier*, 2008 Special Issue.

[Jasmine, 2008] Jasmine (2008). http://www.swarmrobot.org/. *(August 29, 2008)*.

[Kornienko et al., 2007] Kornienko, S., Kornienko, O., Nagarathinam, A., and Levi, P. (2007). From real robot swarm to evolutionary multi-robot organism. *IEEE Press*, pages 1483–1490.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press.

[Mahner and Kary, 1997] Mahner, M. and Kary, M. (1997). What Exactly Are Genomes, Genotypes and Phenotypes? And What About Phenomes? *Academic Press Limited*, pages 55–63.

[Meister, 2007] Meister, E. (2007). Development of CAN-based Computational Cells for Multi Robot Organisms. *Diplomarbeit*, 1025.

[Merkl, 2008] Merkl, R. (2008). *Kann die Genetik einen Stammbaum rekonstruieren?*, pages 83–104. Universitï¿½tsverlag Regensburg.

[Mondada et al., 1993] Mondada, F., Franzi, E., and Ienne, P. (1993). *Mobile robot miniaturisation: A tool for investigation in control algorithms*, pages 501–513. Springer Verlag.

[Multi-robot, 2008] Multi-robot (2008). http://www.symbrion.org/. *(August 29, 2008)*.

[Nedjah and de Macedo Mourelle, 2005] Nedjah, N. and de Macedo Mourelle, L. (2005). *Evolutionary Synthesis of Synchronous Finite State Machines*, pages 103–127. Springer-Verlag Berlin Heidelberg.

[Nolfi and Floreano, 2000] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. The MIT Press.

[Rechenberg, 1965] Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment, Farnborough, U.K., Library Translation No. 1122*.

[Rothermel, 2008] Rothermel, M. (2008). Mechanical design and assembly of a swarm robot with docking capabilities for building multi-robot organisms. *Studienarbeit*.

[S-BOT, 2008] S-BOT (2008). http://www.swarm-bots.org/. *(October 12, 2008)*.

[Schlachter, 2008] Schlachter, F. (2008). Analysis and Implementation of Different Energy Management Strategies for Multi-robot Organisms. *Diplomarbeit*, 2684.

[Schwarzer, 2008] Schwarzer, C. (2008). Investigation of Evolutionary Reproduction in a Robot Swarm. *Diplomarbeit*, 2735.

[Schwefel, 1965] Schwefel, H.-P. (1965). Kybernetische Evolution als Strategie der experimentellen Forschung in der Strï¿½mungstechnik. *Diploma Thesis*.

[Siphonophore, 2008] Siphonophore (2008). http://siphonophores.org/index.php. *(August 29, 2008)*.

[SWARMANOID, 2008] SWARMANOID (2008). http://www.swarmanoid.org/. *(October 12, 2008)*.

[SWARMROBOT, 2008] SWARMROBOT (2008). http://www.swarmrobot.org. *(August 29, 2008)*.

[SYMBRION, 2008] SYMBRION (2008). http://www.symbrion.eu. *(August 29, 2008)*.

[Walker et al., 2008] Walker, J., Garrett, S., and Wilson, M. (2008). Evolving Controllers for Real Robots: A Survey of the Literature. *SAGE Publications*.

[Weicker, 2007] Weicker, K. (2007). *Evolutionï¿½re Algorithmen*. Teubner Verlag.

[WIKIPEDIA, 2008] WIKIPEDIA (2008). http://www.wikipedia.org. *(August 29, 2008)*.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Florian Leiß)