Marshall Plan Scholarship Research Report[1]

Application of the Spherical Induction Motor to Dynamically Stable Robots

Author
Olaf Saßnick

1st. Advisor Prof. Ralph Hollis
Carnegie Mellon University, Pittsburgh, USA

2nd. Advisor Prof. Robert Merz
Salzburg University of Applied Sciences, Salzburg, Austria

Monday 16th November, 2015

[1]The content of this report is based on the authors master thesis with same title.

Abstract

The content of this report is based on the master thesis with same title.

This work gives an introduction to the spherical induction motor and its application to dynamically stable robots [19]. It restricts to mobile balancing robots with a single point of contact to the floor. The robot balances its main body on a single spherical wheel. It moves by leaning towards a direction, and therefore by shifting its center of mass with respect to the point of contact on the floor. This category of robots is known as ballbot. They have been invented by Ralph Hollis in the Microdynamic Systems Laboratory of the Carnegie Mellon University in Pittsburgh, in 2005 [17].

The existing drive mechanisms for a ballbot have always been a trade-off between mechanical complexity and efficiency. The required omni-directional movement capabilities make it challenging to achieve both together: a high efficiency and a low complexity.

The spherical induction motor is considered a potential superior alternative to existing mechanical drive systems for ballbots. Instead of relying on mechanical contact, the torque is applied via a generated electromagnetic field to the main wheel.

Overview

The very first section (1 Introduction) sets out to give the reader a broader overview on dynamic stable robots (ballbot). The motivation behind this non-conventional design approach for mobile robots is explained and possible usage scenarios are given, where conventional mobile robots are outperformed. As well the drawbacks of current ballbot implementations are briefly covered, and why the spherical induction motor poses an alternative worth exploring.

Next, section 2 (Conventional Drive Mechanisms) gives a short overview on existing mechanical drive mechanisms for ballbots, whereas section 3 (Spherical Induction Motor) introduces the function principle of the spherical induction motor ([19]).

As the ballbot, with only one single point of contact to the floor, represents an inherently unstable system, an active balancing control is required. The first basic steps towards an actual balancing control implementation are taken in section 4 (Dynamic Model). Here, a simplified 2D dynamic model of the robot is being introduced and in addition, a 3D simulation in the open dynamics engine (ODE) is attempted for the robots dynamic behavior. Results for scenarios with different start conditions are compared. In section 5 (Control), the balancing control design is attempted based on the dynamic model, which has been introduced beforehand in section 4. A simple full state feedback state-space approach and a conventional approach with cascaded PID and PD controllers are being looked at.

In section 6 (Implementation) focus is placed on the actual implementation of a ballbot with the novel spherical induction motor (named SIMbot).

Finally, in section 7 (Testing and Initial Results) early performance results of SIMbot are presented. This includes the balancing performance, station-keeping and point-to-point motion trajectories.

Contents

List of Figures

List of Tables

Listings

1. Introduction

The robotics industry has been a fast growing branch over the last two decades. In 2003 the number of worldwide yearly delivered industrial robots was 82,000 which climbed to 178,000 units in 2013, see [25]. That equals an increase of $\approx$ 217%. The demand for industrial robots and automation is assumed to continue growing. However, the forecasts do not solely focus on the usage in the industrial sector, where robots already replaced much of the labor work, but also look for future applications of robots closer to the human environment ([27], [31]). New opportunities arise in the market for service robots. While today very much in its beginning, the market for service robots is predicted to emerge as a fast growing market [27]. According to [26]a service robot is a robot that performs useful tasks for humans or equipment excluding industrial automation application. According to [41] already in 2013 the majority of service robots were mobile platforms. However, where can mobile service robots be useful? What kind of applications are suitable for a mobile robot in a human environment? How can a robot assist humans and where can they help and not just hinder us? There is no definite answer yet, however there exist pilot studies for numerous tasks and first service robots have been successfully deployed. Some examples are described here:

- Panasonic Hospital Robot:
  The project started back in 2004. The main idea is to relieve the medical staff at hospitals by automatizing the transport of small objects, for example drugs, from one place to another. To do so, the robot needs to navigate and maneuver safely inside buildings without interfering with obstacles or people [7].

- Care-O-Bot:
  The Care-O-Bot by the German Fraunhofer Society is aimed to take over tasks of a butler or human assistant. It is designed to help people within domestic environment as well for elderly care [14].

- Savioke robot:
  Savioke is a company located in California. For now it produces a mobile delivery

robot specialized for the hospitality industry. However the company plans to extend their fleet of service robots, see [32]. One task for example is to deliver items to hotel rooms.

All three of them are mobile robots, designated to operate in environments used by humans. As a consequence they must be able to deal with humans in a safe and predictable manner, unlike most industrial robots, which work alone locked down in a safety cage. For example, their environment can be cluttered with people.

Nevertheless the mobile service robot should be able to navigate through this cluttered space in a very gentle way, avoiding incidents and adapt quickly to a changing environment. Ideally they would behave very much like a human, moving similar as we do based on our instinct. Their movement pattern should blend in. They should behave as one would expect from a human. To accomplish this, a small footprint and height similar to a human helps.

All three robots are built statically stable and locomote with wheeled drives. To ensure that the robot does not tip over on contact, a low center of gravity is required. Nevertheless, due to the required height, the base is required to be wider than a human body. All three designs keep the robot base as compact as possible, which however compromises the dynamic capabilities of the robots and reduces their stability. Slightly over-exaggerated, but true in essence, the these robots are heavy and slow. It reduces their possible benefits in serving and delivering goods.

However, what are the alternatives? Assuming the environment is built to fit the needs of humans, there exists the belief, that robots have to resemble them closely to achieve best results in terms of flexibility and adaptation. Furthermore, if a robot should maneuver human-like, then it may as well appear, that it should be built like a human, i.e. resemble the human body, as this is what our non-natural surroundings are designed for.

Following this logic, the classic humanoid robot is the prime candidate for operation in human environments. Humanoids resemble the human body closely. In consequence these robots reach the desired height of a human while still having a similar footprint.

Stability is reached by dynamic active balancing control. In theory this approach seems to be ideal. But the benefits come at a cost, just think of the many joints, all of them have to be motorized and controlled! At the DARPA 2015 Challenge, teams from all over the world tried to push the boundaries. Most of them competed with bipedal robots [8].And even though being remotely supervised and operated, only one among the 25 finalists of the 2015 DARPA event did not fall or need physical human intervention like a reset, [4]. To summarize, the current technology for bipedal robots is still a few ”steps” away from performing smooth and robust movements, like quick direction changes, faster walking or even running.

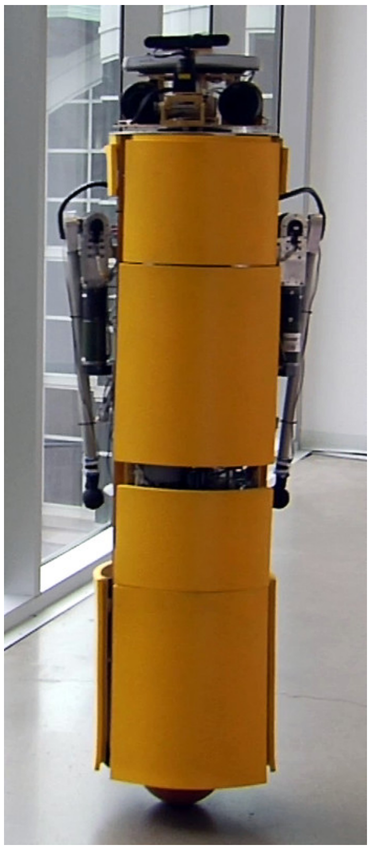
Figure 1: The original ballbot [37]

The ballbot robot category takes a different approach to solve this problem. Instead of resembling the human mechanics/ kinematics, it tries to achieve the same movement capabilities with a different mechanical concept. By balancing the upper body on a single spherical wheel, it is capable of smooth omni-directional movements and able to perform trajectories, which are similar to what humans can achieve. Biggest advantage of this approach is the vastly reduced mechanical complexity. This as well reduces the complexity of the required robust control algorithms and results in a robot with high dynamic stability. Due to the dynamic stability the ballbot can be as tall as a human, and still maintain a small footprint (see Fig. 1). Furthermore this gives the robot an implied physical compliance. The robot can be moved or pushed away with a minimum amount of external force, which leads to increased safety.

Despite of the mechanical simplicity compared to the legs of humanoids or bipedal robots, the omni-directional drive mechanism for a spherical wheel is nevertheless challenging and of critical importance to the operation. So far two different mechanical concept have emerged, the first is the inverse mouse-ball drive, invented by Ralph Hollis [17], while the second uses a set of omni-directional wheels to drive the spherical wheel underneath. Both of them however have individual disadvantages. The inverse mouse-ball drive has lossy friction effects and requires five electric motors to work well, whereas the concept with omni-directional wheels only has limited load-carrying capabilities and requires a large amount of movable parts and at least three electric motors. (see Fig. 2, [5]).

The spherical induction motor is a potential superior alternative to the mechanical drive systems. The application of the spherical induction motor vastly reduces the mechanical complexity, basically reducing the robot to two moving parts, the spherical rotor of the induction motor and the main body of the robot. Instead of requiring mechanical contact, the torque is applied via a generated electromagnetic field.

This work focuses on the driving system and the overall control of a ballbot actuated by a spherical induction motor.

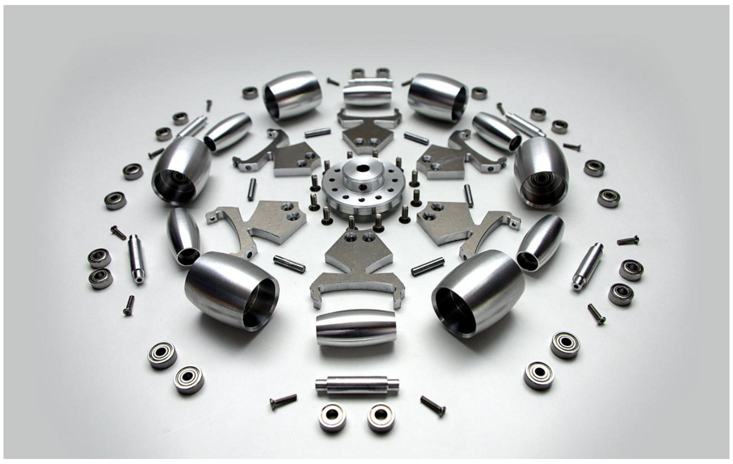

Figure 2: A disassembled omniwheel of the FHS-ballbot. It has 12 separate rollers. Number of parts: 78, [5]

2. Conventional Drive Mechanisms

2.1. Inverse Ball Mouse Drive Mechanism (CMU Ballbot)

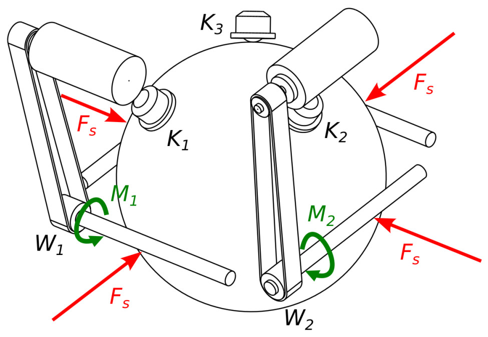

Figure 3: Schematic Overview of the drive system of the CMU ballbot

The first ballbot built at the Carnegie Mellon University in the Microdynamic Systems Laboratory uses an inverse ball mouse drive mechanism to achieve a omni-directional movement. The principle is similar to a mechanical computer mouse, where two encoder discs driven by a ball on the surface ground are used to track the users motion. As the name already implies however, the mechanism has been reversed. The ball is no longer a passive component but instead actuated [1].

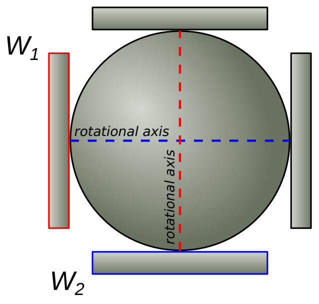

Figure 4: Schematic top view of the CMU ballbot

The principle is shown in Fig. 3. Two blank shafts W_1 and W_2 are driven via electric motors. Each motor is placed at a higher position to allow for smaller overall dimensions. The torque transmission from the motor to the blank shaft is realized with a toothed belt. The two opponent shafts of W_1 and W_2 are passive.

Ball transfer units (depicted in Fig. 3 as K_1, K_2 and K_3) support the main body on the ball. The ball itself is made out of a hollow aluminum sphere, covered with a urethane plastic layer. The aluminum layer helps to keep the shape and can reduce the amount of deform-ability while the outer urethane layer creates a surface with desirable friction properties.

The preloading force prevents slip and makes movement more controllable but also adds friction, see Fig. 3. Ideally there should exist only one point of contact between shaft and ball. As the point of contact lies within the rotational axis for the other drive system these two ideally do not influence each other, see Fig. 4.

A later version of the CMU ballbot has two more drive motors added, acting on the two remaining passive shafts. These changes result in a better controllability and a smoother torque curve.

One flaw of the drive mechanism is that it does not allow a rotation around the z-axis. This is added with a further mechanism, a rotary table, mounted on top of the drive mechanism.

2.2. Omniwheel Mechanism

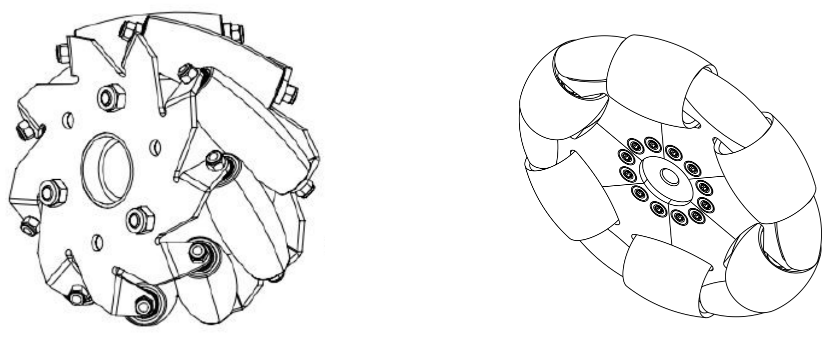

Figure 5: Mecanum wheel [29] (left) and Omniwheel with different radia (right)

The name already suggests it: an omni-wheel is intended to allow movements in more than one direction. Compared to a normal wheel, which given a no slip condition, has only one rotational degree of freedom, the omniwheel adds one more by allowing an additional sideways movement. Two configurations of the omniwheel have proved their worth in practice, shown in Fig. 5:

- Mecanum wheel: To ensure rotation in two directions, the wheels of the Mecanum wheel are turned by a defined inclination angle.

- Omniwheel with different radia: By combining wheels with smaller and larger radia running on a spherical surface can be accomplished with almost no discontinuity [30], [40]. This configuration is used by the Rezero built at the ETH Zurich and the BallIP built at the Tohoku Gakuin University [42], [12].

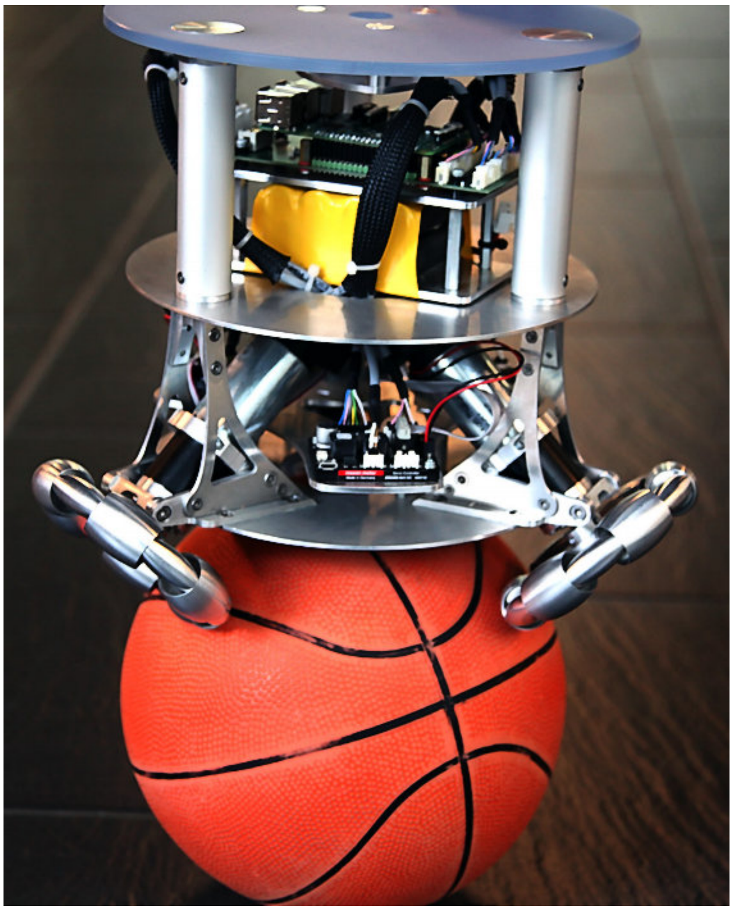

Figure 6: Omniwheels with different radia, as seen on the FHS-ballbot [5]

3. Spherical Induction Motor

This section does briefly cover the spherical induction motor (short: SIM). It has been invented by Ralph Hollis. A description can be also found in the following papers: [3] and [21]. There already existed other spherical motors which however are unable to perform continuous rotation, due to their limited rotation range. Besides that, so far developed spherical motors have not been able to generate a sufficient amount of torque to drive a human-sized robot ([23], [16], [35]). The SIM was developed particularly with the application as an alternative omni-directional drive-system for the ballbot in mind.

3.1. Function Principle

The function principle of the spherical induction motor (SIM) is comparable to a linear induction motor with a single-sided stator, the abbreviation SLIM is commonly used for such kind of motor. The SLIM is classified as a machine with one degree of mechanical freedom, since the movement of the actuated part can be only commanded in one degree of freedom.

A SLIM consists of an armature (the stator) and a secondary part:

- **The stator** is made of several stacked iron layers to reduce eddy current losses. The slots in the stator are used to add wire windings, usually a three-phase winding layout is chosen.

- **The secondary** commonly consists of two layers, one conductive thin (aluminum or copper) sheet on the top, and a larger layer below, made of iron.

Applying a sine wave voltage to the three windings of the stator creates a traveling magnetic field, which induces a voltage into the secondary. The conductive layer of the secondary improves the amount of the induced voltage.

The induced voltage generates an eddy current flow in the secondary, which is strengthened by the lower iron layer. Consequently, the eddy current flow creates a magnetic field in the secondary.

The interaction between the two magnetic fields of the stator and the secondary results in a thrust force and as well a normal force component.

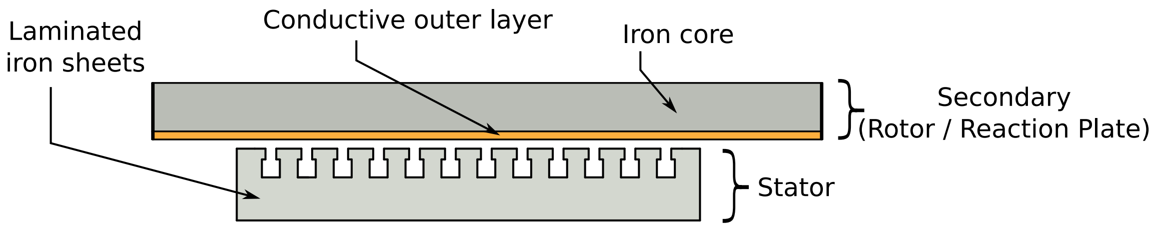

Figure 7: Structure diagram of a single-sided linear induction motor (SLIM)

Taking the function principle of the SLIM and adapting it for a spherical induction motor results in the following changes, see Fig. 8:

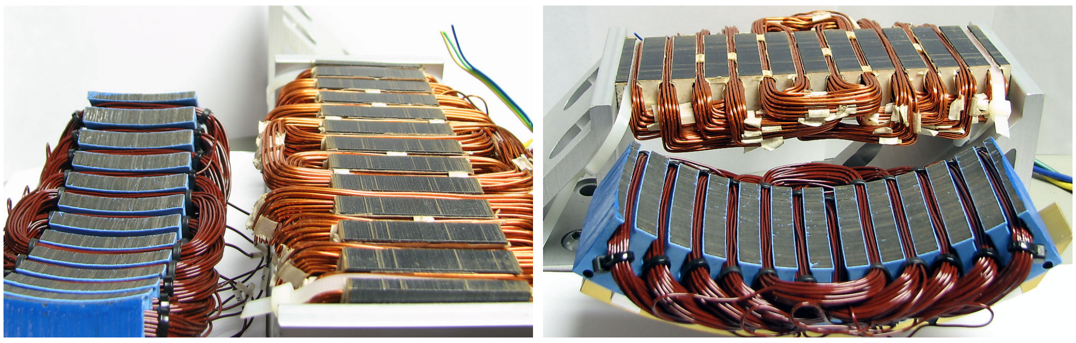

Figure 8: A LIM stator next to a spherical induction motor stator (with blue coating) for comparison

First of all, the shape of the secondary changes. Instead of a plane it becomes a sphere. The outer and the inner layer remain. The larger inner layer is now required as a structural part, to keep the sphere in shape under magnetic forces from the stator in addition to hold the external load added by the robots body [21]. The secondary is supported with omni-directional ball transfer units, to allow for three rotational degree of freedom.

The stator shape is adapted to fit the spherical surface of the secondary. To actuate all three degrees of freedom of the secondary, which now is a rotor, a minimum of $n_s = 3$ stator elements are required. The thrust force of each stator no longer results in a linear movement of the secondary, instead a rotational movement is introduced. The thrust force of each stator results in a torque, acting on the secondary. The thrust force mapping to a resulting torque vector is covered in section 3.2. Each stator is controlled individually with a three phase circuit electronics board.

3.2. Torque Output

The desired input for the spherical drive should be a torque vector τ_s. However it is only possible to command the generated force acting on the sphere by each stator. Thus a conversion from the resulting torque vector τ_s to n_s single stator output forces is required, as described in [3]. More details regarding this can be found in Appendix B.

3.3. Implementation

The 2015 revision of the SIM is built with of six stator coils, whereas a previous version only had four of them [21]. Each stator coil is driven by a separate controller. More details about the driver board electronics can be found in section 3.2 . Each SIM stator has three phase windings: A, B and C. They are connected in delta mode. The rotor is constrained by six ball transfer units. The torque output of the newer 2015 revision SIM is specifically tuned for balancing mobile robots. The stator coils are positioned to generate a smaller torque around the z-axis in upper body frame and a higher torque around an arbitrary axis passing through the origin, perpendicular to the z-axis (see Fig. 9). In the current state of development, the main focus is put on the dynamic stability of the ballbot. Hence, not having sufficient torque available to perform a rotation around the z-axis is acceptable as does not cause instability. However not being capable to apply the required torque to recover from a steep lean angle is critical. It results in the robot failing to maintain balance, falling over and causing possible damage to itself and its surrounding.

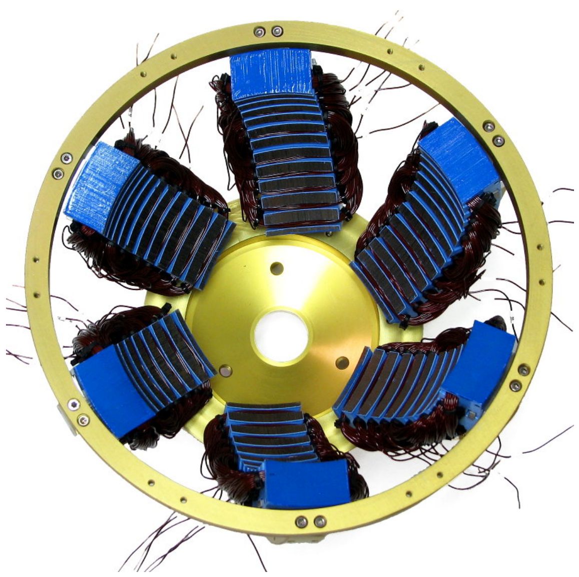
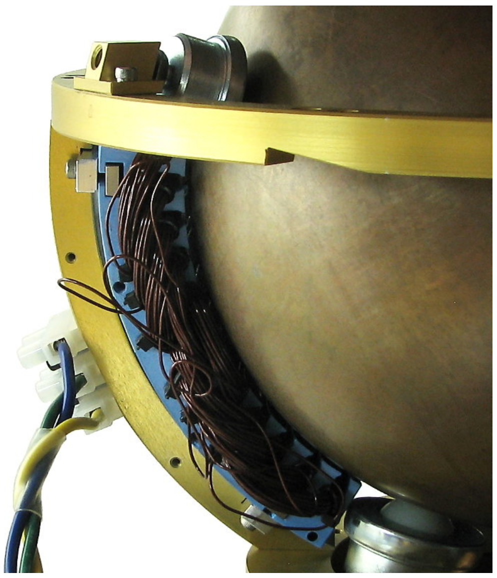

Figure 9: SIM stator frame (left) and stator frame with rotor, constrained by ball transfer units (right)

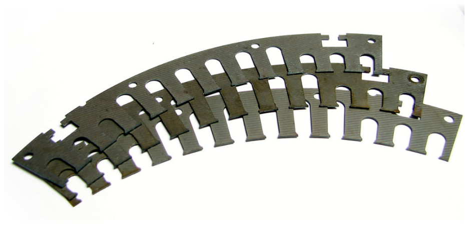

Figure 10: The stators are made out of laser-cut iron sheets

Fig. 11 displays the resulting stator forces, when commanding $\tau_{s.x} = [1, 0, 0]^T$ Nm, $\tau_{s.y} = [0, 1, 0]^T$ Nm, $\tau_{s.xy} = [1, 1, 0]^T$ Nm and $\tau_{s.z} = [0, 0, 0.1]^T$ Nm respectively.

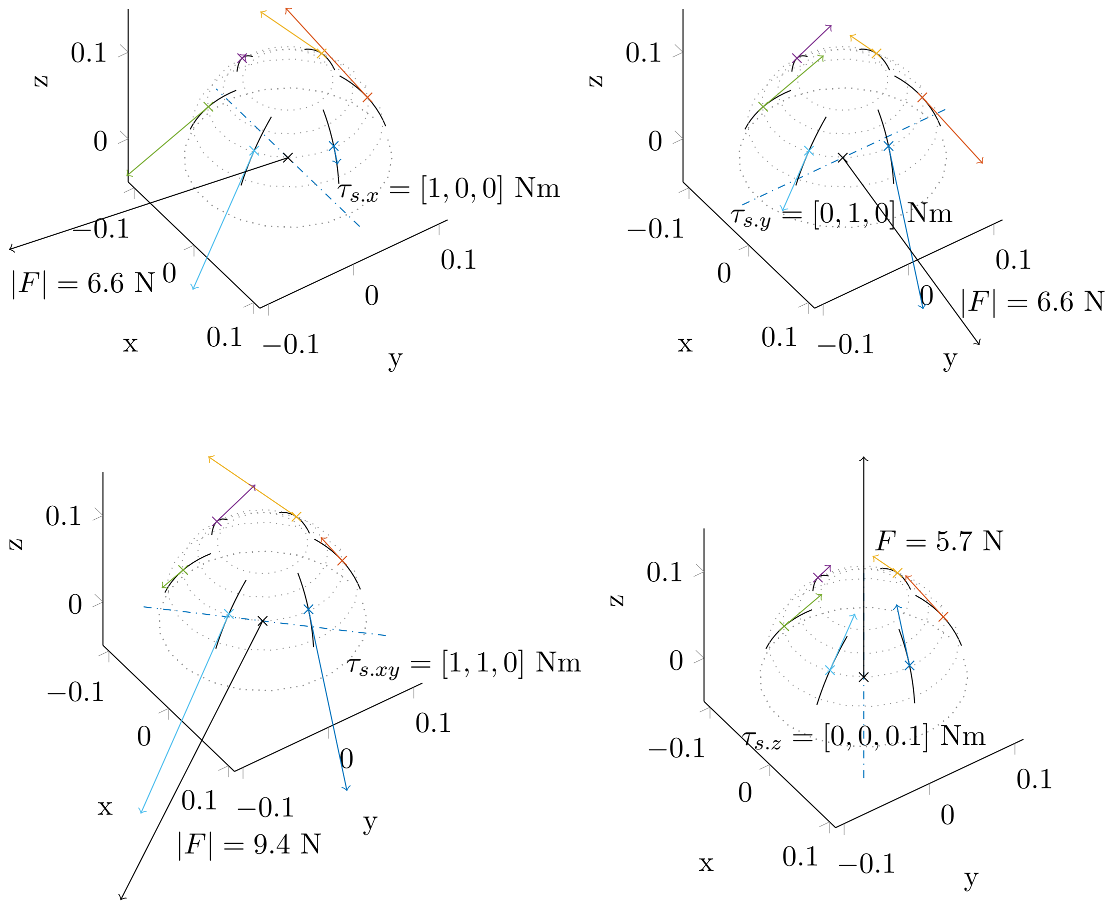

Figure 11: Resulting stator forces for different commanded torque vectors

The individual stator positions are symbolized with arcs in black color, whereas the resulting torque axis is drawn as dashed-dotted blue line. Additionally to the desired torque, a non-zero net force is introduced, acting between the stator frame and the rotor. As expected, for the $\tau_{s.z}$ vector, which applies solely torque around the z-axis, all stators output the same amount of force.

3.3.1. Odometry Sensors

PC-mouse optical sensors are used to measure the angular velocity ω_s and the traveled distance of the spherical rotor. Each sensor i gives readings on two perpendicular axes, x_{mi} and y_{mi} (see Fig. 12). Multiple sensors have to be used to calculate the angular velocity vector ω_s from the sensed surface velocities. According to [20], at least two sensors are required. In principle, the required calculation is similar to the previously discussed torque output mapping and has been shown in [21] and [3]. However as the latest published paper [3] included a slightly incomplete sensor matrix, it seems reasonable to briefly introduce a corrected version (see Appendix B.2).

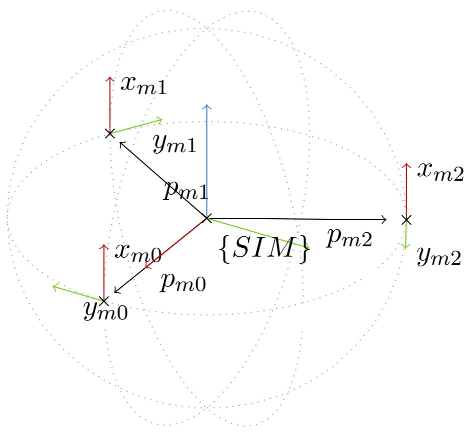

Figure 12: Sensor positions for $n_m = 3$

4. Dynamic Model

Before the actual balance controller design is being discussed, the motion characteristics have to be derived. To achieve this we will derive equations of motion with the Euler-Lagrange approach.

Instead of deriving the equations for a full 3D model, a simplified 2D approach is chosen (see Fig 13). This is referred to as the simplified planar model. To achieve control of the real robot, two planar models, in xz and yz are combined. The combined planar models do not fully represent the dynamics of the actual ballbot, but they give a good enough approximation for smaller lean angles.

The derivation of simplified planar models can be found in [37], the simplified planar model is displayed in Fig. 13.

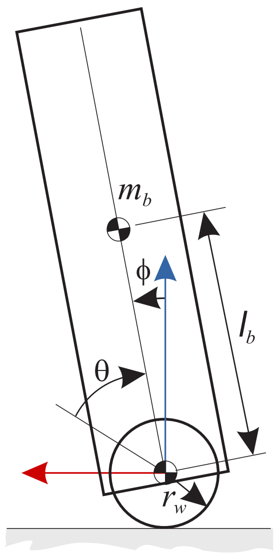

Figure 13: Simplified planar model, adopted from [37]

The global frame origin is assumed to be in the initial wheel center point. A reduction to the two bodies is done:

- Main body b
- Wheel w

A full step-by step solution calculated with the open-source symbolic mathematics tool Maxima is listed in appendix D.

4.1. Parameters

All parameters related to the main body are denoted with the index b:

m_b	Mass
l_b	Distance from wheel center to main body mass center
I_b	Inertia

All parameters describing the wheel are denoted with the index w:

m_w	Mass
r_w	Radius
I_w	Inertia

The gravity of Earth is denoted by g.

4.2. Assumptions

The following assumptions are made to further reduce the complexity of the planar model:

- There exists no slip between the wheel w and the ground

- The floor is parallel to the xy-plane

4.3. Generalized Coordinates

In the simplified planar model all possible states can be described with $n = 2$ coordinate variables, both are indicated in Fig. 13.

- Lean angle ϕ
- Wheel angle θ

Thus, the vector q with generalized coordinates can be described as:

$$q = \begin{pmatrix} \theta \\ \phi \end{pmatrix} \tag{4.1}$$

The lean angle ϕ defines the inclination of the body compared to the vertical (z-direction). Instead of defining the wheel angle θ only with respect to a global reference point on the horizontal axis (x- or y-axis) as done in a few previous works (see [22] and [39]), here θ describes the rotation of the wheel also with respect to the lean angle ϕ and a global reference point. As it is assumed that there is no slip (see section 4.2), the wheel angle describes the distance to the horizontal reference point by a given lean angle ϕ, it can adopt values between $\pm\infty$. This resembles the actual situation on the robot, where the sensors are attached to the main body to collect wheel odometry data, and therefore change their position with respect to the lean angle ϕ.

First, the position vectors for the body b and the wheel w are defined:

$$v_w = \begin{pmatrix} r_w\ (\theta + \phi) \\ 0 \\ 0 \end{pmatrix} \qquad v_b = v_w + l_b \cdot \begin{pmatrix} \sin(\phi) \\ 0 \\ \cos(\phi) \end{pmatrix} \tag{4.2}$$

Next, the corresponding velocity vectors, $\dot{v}_w$ and $\dot{v}_b$, can be derived:

$$\dot{v}_w = \begin{pmatrix} r_w \left(\dot{\theta} + \dot{\phi}\right) \\ 0 \\ 0 \end{pmatrix} \qquad \dot{v}_b = \dot{v}_w + l_b \cdot \begin{pmatrix} \cos(\phi)\ \dot{\phi} \\ 0 \\ -\sin(\phi)\ \dot{\phi} \end{pmatrix} \tag{4.3}$$

Lastly, the angular velocity vectors ω_w and ω_b for both bodies are defined:

$$\omega_w = \begin{pmatrix} 0 \\ \dot{\theta} + \dot{\phi} \\ 0 \end{pmatrix} \qquad \omega_b = \begin{pmatrix} 0 \\ \dot{\phi} \\ 0 \end{pmatrix} \tag{4.4}$$

4.4. Potential Energy

Next, the potential energies of the two bodies are deduced. As the floor is assumed to be horizontal and perfectly flat, the potential energy of the wheel V_w remains constant.

$$V_w = 0 \tag{4.5}$$

The potential energy of the body, denoted as V_b, changes corresponding to the lean angle ϕ:

$$V_b = m_b \cdot g \cdot l_b \cos(\phi) \tag{4.6}$$

4.5. Kinetic Energy

To find the kinetic energy for each body a generalized approach is taken:

$$T = \frac{1}{2} m \ \langle \dot{v}, \dot{v} \rangle + \frac{1}{2} I \ \langle \omega, \omega \rangle \tag{4.7}$$

T represents the objects kinetic energy, $\dot{v}$ its velocity vector and ω its angular velocity vector. The constant m denotes the objects mass property and I its inertia.

With Eq.(4.7) and the further vectors defined before (see 4.3) we can now calculate the kinetic energy of the wheel T_w and the main body T_b.

$$T_w = \frac{(m_w\, r_w{}^2 + I_w)\, \left(\dot{\theta} + \dot{\phi}\right)^2}{2} \tag{4.8}$$

$$T_b = \frac{m_b\, r_w{}^2\, \dot{\theta}^2 + 2\, m_b\, r_w\, \left(l_b \cos\left(\phi\right) + r_w\right)\, \dot{\phi}\, \dot{\theta} + \left(2\, l_b\, m_b\, r_w \cos\left(\phi\right) + m_b\, r_w{}^2 + l_b{}^2\, m_b + I_b\right)\, \dot{\phi}^2}{2} \tag{4.9}$$

4.6. External Forces and Torque

The vector Q represents all of the external/non conservative forces and torques. It is the resultant of all forces and torques acting from outside on the system, modifying the total energy of the system. These can be frictional forces, damping forces or general forces/torques acting from outside on the ballbot.

$$Q = \sum_i F_i + \sum_j \tau_j \tag{4.10}$$

Q has to be written in the selected generalized coordinates. Force F_c and torque τ_c have to be transformed from Cartesian coordinates to the generalized coordinates q first.

$$F = \left(\frac{\partial r_c}{\partial q}\right)^T \cdot F_c \quad , \qquad \tau = \left(\frac{\partial \omega_c}{\partial q}\right)^T \cdot \tau_c \tag{4.11}$$

The vector r_c represents the contact point of the force F_c on the body and ω_c the angular velocity of the body on which the torque τ_c is acting. Both are written in Cartesian coordinates.

The only external force and torque that has to be factored in is the torque τ acting on the wheel w generated by the spherical induction motor. As the stator of the spherical drive is attached to the main body, every torque acting on the wheel w creates an equal counter torque acting on the main body b (Newton's third law, actio = reactio). This is written in Cartesian coordinates:

$$\tau_w = -\tau_b = \begin{pmatrix} 0 \\ \tau_{sim} \\ 0 \end{pmatrix} \tag{4.12}$$

Taking the partial derivatives of ω_w and ω_b with respect to the generalized coordinates q yields the required transformation matrices:

$$J_w = \left(\frac{\partial \omega_w}{\partial q}\right) = \begin{pmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{pmatrix} \qquad J_b = \left(\frac{\partial \omega_b}{\partial q}\right) = \begin{pmatrix} 0 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix} \tag{4.13}$$

At last we need to resolve Q as shown before in Eq.(4.10):

$$Q = (J_1)^T \cdot \tau_w + (J_2)^T \cdot (-\tau_b) = \begin{pmatrix} \tau_{sim} \\ 0 \end{pmatrix} \tag{4.14}$$

Further external forces are neglected for now, but Appendix C gives an example how they can be included.

4.7. Equations of Motion

The Euler-Lagrange approach, as seen in the following Eq.(4.15) , is used to derive the equations of motion, compare [11].

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{q}}\right) - \left(\frac{\partial T}{\partial q}\right) + \left(\frac{\partial V}{\partial q}\right) = Q \tag{4.15}$$

where T is the sum of all kinetic energy in the system, V is the sum of all potential energy in the system and Q represents all external forces and torques.

For the simplified planar system of the ballbot T and V respectively are:

$$T = T_w + T_b \qquad V = V_w + V_b \tag{4.16}$$

The equation (4.9) was already getting quite spacious. So before moving on, a few substitutions are introduced for constant parameter products and sums by looking at Eq.(4.6), Eq.(4.8) and Eq.(4.9) (as done in [37]):

$$\alpha = m_b\, r_w^2 + m_w\, r_w^2 + I_w \tag{4.17}$$

$$\beta = l_b\, m_b\, r_w \tag{4.18}$$

$$\gamma = I_b + l_b^2\, m_b \tag{4.19}$$

$$\epsilon = m_b \cdot g \cdot l_b \tag{4.20}$$

With all substitution rules applied, the total kinetic energy T in the system is:

$$T = \frac{1}{2}\left(\alpha\left(\dot{\theta}\right)^2 + \left(2\,\beta\cos\left(\phi\right) + 2\,\alpha\right)\left(\dot{\phi}\,\dot{\theta}\right) + \left(2\,\beta\cos\left(\phi\right) + \alpha + \gamma\right)\left(\dot{\phi}\right)^2\right) \tag{4.21}$$

Taking the derivatives of T as required for Eq.(4.15) results in:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{q}}\right) - \left(\frac{\partial T}{\partial q}\right) = \begin{pmatrix} \alpha\left(\ddot{\theta}\right) + \left(\beta\cos\left(\phi\right) + \alpha\right)\left(\ddot{\phi}\right) - \beta\sin\left(\phi\right)\left(\dot{\phi}\right)^2 \\ \left(\beta\cos\left(\phi\right) + \alpha\right)\left(\ddot{\theta}\right) + \left(2\,\beta\cos\left(\phi\right) + \alpha + \gamma\right)\left(\ddot{\phi}\right) - \beta\sin\left(\phi\right)\left(\dot{\phi}\right)^2 \end{pmatrix} \tag{4.22}$$

The same is repeated for the V term:

$$\left(\frac{\partial V}{\partial q}\right) = \begin{pmatrix} 0 \\ -\epsilon\sin\left(\phi\right) \end{pmatrix} \tag{4.23}$$

There exists an alternative more compact notation form for the Euler-Lagrange equation, which is hereby introduced, compare [11]. M is usually denoted as the mass matrix with shape n by n, whereas n is the number of coordinates. C is the centrifugal and coriolis forces vector and G is the vector of gravitational forces.

$$M(q) \cdot \ddot{q} + C(q, \dot{q}) + G(q) = Q \tag{4.24}$$

As only Eq.(4.22) contains the second derivative of q, M can be expressed by only taking coefficients from Eq.(4.22). This results in:

$$M(q) = \begin{pmatrix} \alpha & \beta \cos(\phi) + \alpha \\ \beta \cos(\phi) + \alpha & 2\,\beta \cos(\phi) + \alpha + \gamma \end{pmatrix} \tag{4.25}$$

If the mass matrix M is given, the vector C yields:

$$C = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{q}}\right) - \left(\frac{\partial T}{\partial q}\right) - M\begin{pmatrix} \ddot{\theta} \\ \ddot{\phi} \end{pmatrix} = \begin{pmatrix} -\beta \sin(\phi) \left(\dot{\phi}\right)^2 \\ -\beta \sin(\phi) \left(\dot{\phi}\right)^2 \end{pmatrix} \tag{4.26}$$

No expression has to be rewritten to retrieve G. The vector G already equals $(\partial V/\partial q)$ of Eq.(4.23):

$$G = \left(\frac{\partial V}{\partial q}\right) = \begin{pmatrix} 0 \\ -\epsilon \sin(\phi) \end{pmatrix} \tag{4.27}$$

4.8. Linearisation

The linearisation is done with the first order of the Taylor series approximation. First, an equilibrium point has to be found for the system. The equilibrium point is a system operating point, in which a specific equilibrium input does not change the system state. So, given there are no external disturbances, the system will remain in that operating point for all time.

For the planer model of the ballbot an equilibrium point is found by applying the following conditions: It is assumed that the robot initially has a zero lean angle and no body velocity. Furthermore the robot is assumed to be positioned in the world coordinate system origin and the wheel should have no angular velocity. Under these conditions no torque has to be applied to the wheel (system input) at any time.

Formulated in general coordinates (see Eq.(4.1)), these assumptions for the operating point are:

$$q_0 = \begin{pmatrix} \theta_0 \\ \phi_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \dot{q}_0 = \begin{pmatrix} \dot{\theta}_0 \\ \dot{\phi}_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{4.28}$$

The formula for the first order Taylor approximation is, where $t(x)$ is a non-linear term, and x_0 the selected equilibrium value for variable x [2].

$$t^*(x) \approx t(x_0) + t'(x_0) \cdot (x - x_0) \tag{4.29}$$

All linear terms remain unchanged, while all non-linear terms are approximated around the value x_0 with a linear gradient. Eq.(4.30), Eq.(4.31) and Eq.(4.32) can be deduced from the taylor-approximation Eq.(4.29) in the chosen equilibrium point (see Eq.(4.28)). Eq.(4.24) from the previous Section 4.7 is then taken as a basis for the linearisation. The substitions are performed according to Eq.(4.30), Eq.(4.31) and Eq.(4.32).

$$\cos(\phi) \approx 1 \tag{4.30}$$

$$\sin(\phi) \approx \phi \tag{4.31}$$

$$(\dot{\phi})^2 \approx 0 \tag{4.32}$$

Next, the linearized version of Eq.(4.24) can be formulated:

$$\underbrace{\begin{pmatrix} \alpha & \beta + \alpha \\ \beta + \alpha & 2\beta + \alpha + \gamma \end{pmatrix}}_{M^*} \cdot \ddot{q} + \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{C^*} + \underbrace{\begin{pmatrix} 0 \\ -\epsilon\,\phi \end{pmatrix}}_{G^*} = \underbrace{\begin{pmatrix} \tau_{sim} \\ 0 \end{pmatrix}}_{Q_{NC}} \tag{4.33}$$

4.9. State Space Representation

In a state space representation, a system of two equations is commonly used. The first equation determinates the system state, the second updates the systems output:

$$\dot{x}(t) = A\,x(t) + B\,u(t) \tag{4.34}$$

$$y(t) = C\,x(t) + D\,u(t) \tag{4.35}$$

The linearized model, from Eq.(4.33), is taken as a basis for the state space representation. The two equations are solved for the highest order derivatives, $\ddot{\theta}$ and $\ddot{\phi}$ [2]:

$$\ddot{\theta} = -\frac{(\beta+\gamma)\,T_{sim} - \beta\,\epsilon\,\phi}{\beta^2 - \gamma\,\alpha} \tag{4.36}$$

$$\ddot{\phi} = \frac{(\beta+\alpha)\,T_{sim} - \alpha\,\epsilon\,\phi}{\beta^2 - \gamma\,\alpha} \tag{4.37}$$

Next, the system state vector $x(t)$ and system input u are chosen:

$$x(t) = \begin{pmatrix} \theta \\ \phi \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} \qquad u = \tau_{sim} \tag{4.38}$$

The system matrix A and the input weight matrix B are derived from Eq.(4.36):

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-\beta\,\epsilon}{\beta^2-\gamma\,\alpha} & 0 & 0 \\ 0 & \frac{-\alpha\,\epsilon}{\beta^2-\gamma\,\alpha} & 0 & 0 \end{pmatrix} \qquad B = \frac{1}{\beta^2 - \gamma\,\alpha} \begin{pmatrix} 0 \\ 0 \\ \beta+\gamma \\ \beta+\alpha \end{pmatrix} \tag{4.39}$$

Next, the output matrix C and the feed forward matrix D for the system output $y(t)$ are

[2] ε is substituted, see Eq.(4.20)

determined:

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.40}$$

C is an identity matrix, as all components of the state vector $x(t)$ can be measured with sensors on the robot, and therefore used as controller input.

4.10. Stability of the linear system

To evaluate the stability of the system, the poles of system matrix A have to be determined. These are the solutions to the characteristic polynomial. The solution is simply a linear combination of Exponentials raised to the eigenvalues of the system matrix A. If all exponentials are raised to a negative value, the system is decaying and therefore stable, while a positive exponent results in an unstable system.

Solving A for eigenvalues,

$$\det(A - I\lambda) = 0 \tag{4.41}$$

yields:

$$\lambda_{1,2} = 0 \tag{4.42}$$

$$\lambda_{3,4} = \pm\sqrt{\frac{-\epsilon \cdot \alpha}{\beta^2 - \gamma\,\alpha}} = \sqrt{\frac{-\epsilon \cdot \alpha}{\beta^2 - \gamma\,\alpha}} \tag{4.43}$$

Two poles are located in the origin, while the third and the fourth require further investigation. If the following condition (in Eq.(4.44)) can be assured, the poles created by $\lambda_{3,4}$ do not form an imaginary pole pair:

$$\frac{-\epsilon \cdot \alpha}{\beta^2 - \gamma\,\alpha} \geqq 0 \tag{4.44}$$

Substituting α, β and γ with their original expressions (from Eq.(4.17), Eq.(4.18) and

Eq.(4.19)) and factoring out shared terms results in:

$$\frac{l_b\ (m_b\ (g\, m_w\, r_w{}^2 + g\, i_w) + g\, m_b{}^2\, r_w{}^2)}{l_b{}^2\, m_b\ (m_w\, r_w{}^2 + i_w) + i_b\ (m_w\, r_w{}^2 + i_w) + i_b\, m_b\, r_w{}^2} \geqq 0 \tag{4.45}$$

We can see in Eq.(4.45) that the expression underneath the radical sign cannot become negative as the length l_b, the earth gravity g, the masses m_w & m_b, the radius r_w, and i_w & i_b are all positive physical constants.

To summarize, all poles only have a real part. Therefore the system does not exhibit oscillation in a step or impulse response. However, as one pole is located in the positive half on the real axis, the system must be considered unstable.

This should not come as a surprise, given that there only exists one contact point with the floor, whereas to be statically stable, a minimum of three contact points forming a triangular profile are required.

4.11. Comparison with the non-linear system

Without doubt the linearized system is easier to work with, but how accurate is it compared to the original non-linear system?

Under normal conditions, the lean angle stays close to zero, a lean angle larger than 5-10 deg. is unusual. For example, during acceleration the original ballbot does not exceed a max lean angle ϕ of 5 deg [33]. As such, a close approximation of the linearized system is only considered to be required for $\phi < 10$ deg.

Finding solutions for the non-linear system is attempted by applying numerical methods. For example, the Euler or other Runge-Kutta methods can be used to find approximative solutions for ordinary differential equations. In order to apply them, the system has to be rewritten first:

$$\dot{x(t)} = f(x(t), t) \qquad \text{for initial condition } x_0 \text{ at } t_0 \tag{4.46}$$

x is the system state vector and f is a continuous function depending on x and t. Given the initial system condition x_0, a solution can be approximated.

Eq.(4.46) is not much different from the first state space equation, Eq.(4.34). However due to the non-linear nature, having a separate input matrix B and a system matrix A is not feasible. The state variables vector x remains unchanged compared to the linear system (see Eq.(4.38)):

$$x(t) = \begin{pmatrix} \theta \\ \phi \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} \tag{4.47}$$

Solving the non-linear system from Eq.(4.24) for $\ddot{\theta}$ and $\ddot{\phi}$ yields:

$$\ddot{\theta} = \frac{-\left(2\,\beta\cos\left(\phi\right)+\alpha+\gamma\right)\,T_{sim}-\beta\,\left(\beta\cos\left(\phi\right)+\gamma\right)\,\sin\left(\phi\right)\,\left(\dot{\phi}\right)^2+\epsilon\,\left(\beta\cos\left(\phi\right)+\alpha\right)\,\sin\left(\phi\right)}{\beta^2\cos\left(\phi\right)^2-\gamma\,\alpha} \tag{4.48}$$

$$\ddot{\phi} = \frac{\left(\beta\cos\left(\phi\right)+\alpha\right)\,T_{sim}+\beta^2\cos\left(\phi\right)\,\sin\left(\phi\right)\,\left(\dot{\phi}\right)^2-\alpha\,\epsilon\sin\left(\phi\right)}{\beta^2\cos\left(\phi\right)^2-\gamma\,\alpha} \tag{4.49}$$

The denominator on the right hand side is the same for both equations. It contains the state variable ϕ. Hence, a non-zero denominator has to be ensured for the range of $\cos(\phi)$, which is true given the following condition:

$$\beta^2\cos\left(\phi\right)^2-\gamma\,\alpha\neq 0 \quad\longrightarrow\quad -\frac{\sqrt{\gamma\,\alpha}}{\beta}<-1 \text{ or } \frac{\sqrt{\gamma\,\alpha}}{\beta}>1 \tag{4.50}$$

Using the parameters from ... the condition formulated in Eq.(4.50) is met. The denominator always remains non-zero.

With the state vector formulated in Eq.(4.47) and the explicit results for the second derivatives $\ddot{\theta}$ and $\ddot{\phi}$ it is now possible to solve the non-linear system numerically.

The MATLAB® implementation of the Runge-Kutta method, `ode45`, is used to solve the differential equations system. The numerical calculation is stopped when one of the following criteria is met:

$$\phi > 90^\circ \quad \text{or} \quad \phi < -90^\circ \tag{4.51}$$

In both cases the lean angle ϕ indicates, that the robot has fallen and is unable to recover.

Listing 1 shows the implemented MATLAB® function to calculate the state vector x over time, given an initial state x_0 and input u. The passed argument `ssEq` is a function handle to a custom function. This function is called from `ode45` each timestep and returns the calculated derivative $\dot{x}$ (as solved in Eq.(4.49) and Eq.(4.48)) for a given a state vector x and system input u. Both return all state vector values over time.

```
function [xv,t] = nonLinStateSpaceSim(ssEq, x0, uv, Ts, tEnd)
    t = 0:Ts:tEnd;
    xv = zeros(4,length(t)); xv(:,1) = x0;
    for i = 2:length(t)
         [to,xo] = ode45(ssEq, [t(i)-Ts t(i)],[xv(:,i-1); uv(i-1)]);
         xv(:,i) = xo(end,1:4)';
␣␣␣␣end
end
```

Listing 1: Matlab function, calculating the nonlinear system response

Listing 2 shows the custom MATLAB® function to calculate the linear system response. The function expects a discrete state space system. The continuous linearized system is discretized with the `c2d()` function (zero order hold, sampling time $T_s = 1/200$ s) beforehand.

```
function [xv,t] = discreteLinStateSpaceSim(dsys, x0, uv, tEnd)
    t = 0:dsys.Ts:tEnd;
    xv = zeros(4,length(t)); xv(:,1) = x0;
    for i = 2:length(t)
        if (abs(xv(2,i-1)) >= pi/2)
            xv(:,i) = xv(:,i-1); % freeze
            continue
        end
        xv(:,i)= dsys.A*xv(:,i-1) + dsys.B*uv(:,i-1);
    end
end
```

Listing 2: Matlab function, calculating the linear discrete system response

The behaviors of the linear and non-linear system are compared in two scenarios:

1. **Initial non-zero lean angle**

   The robot is put in an initial state x_0, where the lean angle $\phi = 0.1$ deg. The system input T_{sim} is set to zero.

2. **Input step**

   Applying an input step to the system is done by setting $T_{sim} = 1$ Nm. This equals commanding 1 Nm of torque to the drive system.

Fig. 14 shows the resulting wheel travel θ and lean angle ϕ over time for an initial non-zero lean angle. The robot body starts falling over in the positive direction on the X-axis whereas the wheel moves backwards in the opposite direction.

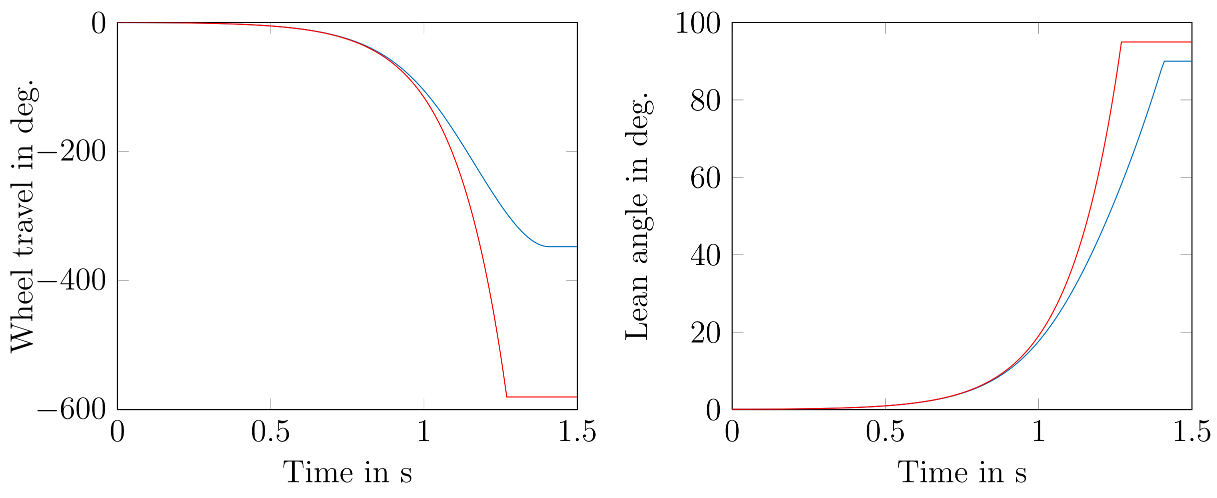

Figure 14: Comparison between the linear (red) and non-linear system (blue) response, given a initial non-zero lnean angle ($\phi = 0.1$ deg)

Fig. 15 displays the resulting wheel travel θ and lean angle ϕ over time with an input step applied at $t = 0$. The wheel starts moving in positive direction on the X-axis, while the robot body starts falling over in the opposite direction.

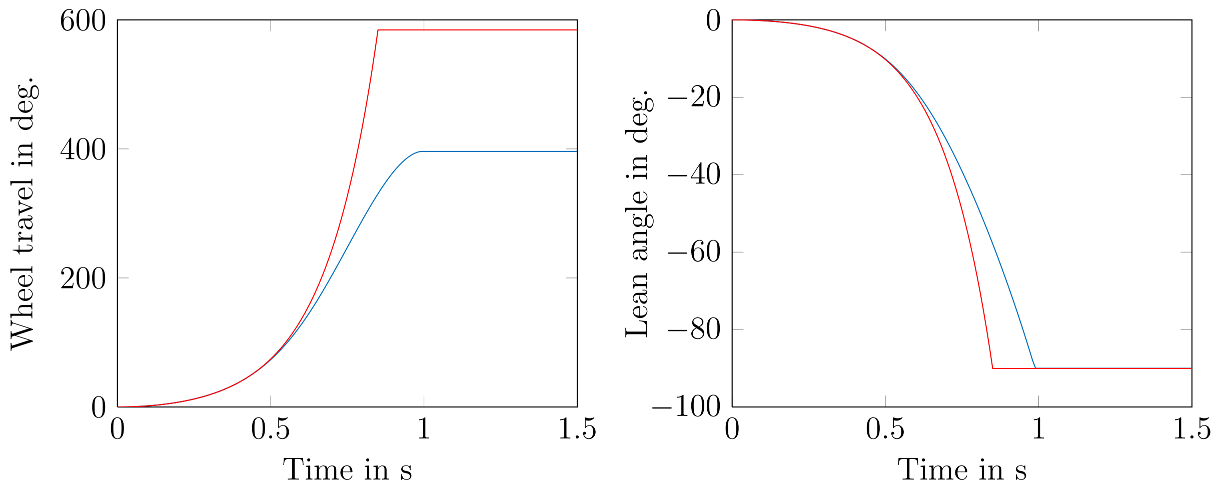

Figure 15: Input step at $t = 0$, comparison between the linear (red) and non-linear system (blue) response

Fig. 16 and 17 give a more detailed view on the error caused by the linearisation. The vertical dashed lines indicate where the non-linear system response reached a lean angle ϕ of $5°$, $10°$ and $45°$ respectively. For both scenarios the error increases depending on the lean angle ϕ. The linearized system response becomes less accurate for a larger ϕ due to the small angle approximation. In both figures the wheel travel error increases with a higher rate than the lean angle error. However, for $\phi < 5°$, the response of the non-linear system is approximated with negligible error. At $\phi = 10°$ the lean angle error is 3.96 and 1.28 percent respectively.

Given these results it seems acceptable to design a controller based on the linearized system.

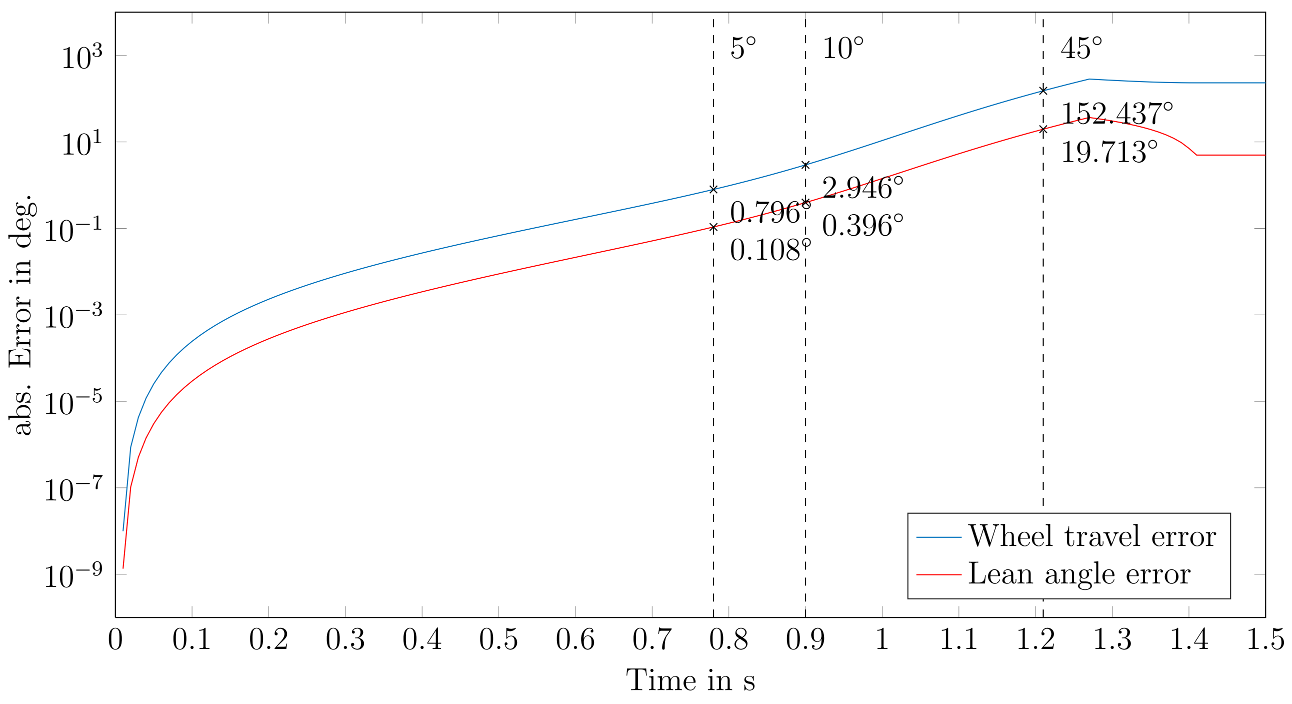

Figure 16: Error due to linearisation - scenario 1: Initial non-zero lean angle (logarithmic scaling on y-axis)

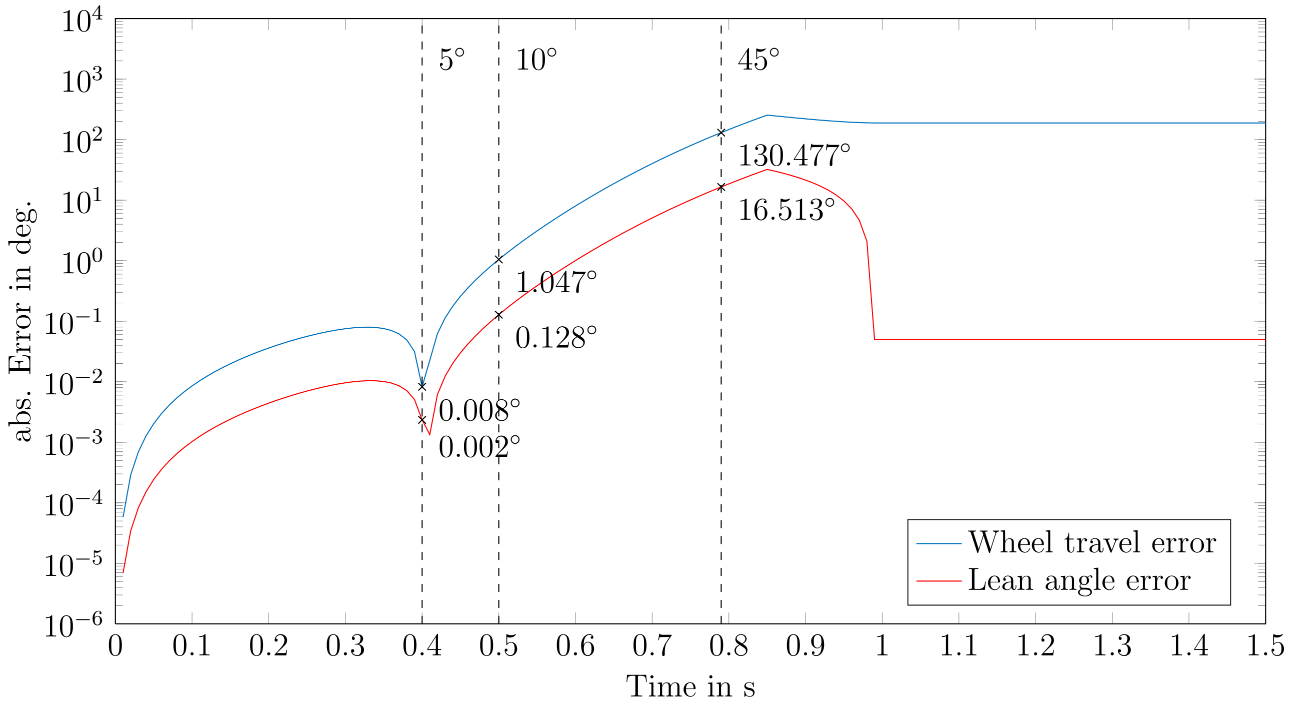

Figure 17: Error due to linearisation - scenario 2: Input step (logarithmic scaling on y-axis)

4.12. Simulation with a physics engine

This section describes how to implement the robot in a physics engine to simulate the motion dynamics. ODE [3] is used to accomplish this task. This engine is open source software and has been available since 2001 [9]. It has been successfully used in many robotics simulation software already, most notably in the DARPA Humanoid Challenge simulator in 2015, see [8].

ODE applies the Euler method to calculate constrained rigid body dynamics for each simulation time step. The implemented algorithms are intended for real-time applications, which means the solution precision is reduced in favor of a lower computation time. For example, instead of repeating an iterative procedure until a certain convergence limit has been reached, only a fixed amount of iteration steps are being carried out.

The previous section already described an approach how to work with the more accurate non-linear system dynamics equations, see 4.11. What additional gains can be expected from a simulation with a physics engine? First off, it provides a way to counter-check the plausibility of the derived system equations from the previous section, as no knowledge about them is required to build the simulation in ODE. And secondly, a model in 3D-space can be obtained this way with little effort, while up to this point, only the planar model has been discussed. This can be useful at a later stage for testing the actual controller for the real robot.

4.12.1. Preparation

At the beginning a scene has to be set up. This is done by positioning bodies and joints in a dynamics world object, the `ODEworld`.

The robot is modeled with two solid bodies: one body with spherical shape represents the wheel, while the upper body is abstracted as a body with a rectangular solid shape.

[3]open dynamics engine

Each body is associated with mass, inertia and a collision shape. Regarding the collision shapes, it is important to take special care, that bodies do not intrude each other in their initial position. Otherwise the simulation will give inconsistent results.

A ball-socket joint is used to model the connection between the upper body and the wheel. The joint is positioned in the center of the wheel, as can be seen in Fig. 18. The rotation of the ball-socket joint is internally stored in quaternions, therefore no gimbal lock phenomena should occur.

In addition, an angular motor joint is created in the same position. The angular motor is configured for three rotational degrees of freedom (to generate torque around three axes), where each axis is defined relative to the upper body frame of the robot. During simulation, the `dJointAddAMotorTorques()` function of the angular motor is used to simulate the drive torque input. This function call first transforms the commanded torque vector into the global frame. Then the torque is added to the two bodies linked to the angular motor, whereas the torque added to the second body is reversed.

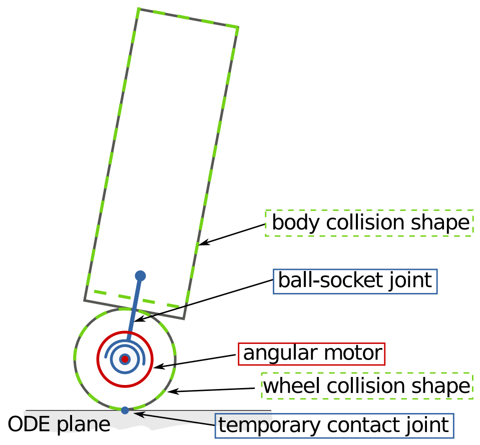

Figure 18: ODE model, showing the collision shapes and joint placements

The robot has now been modeled and is ready to roll, the last thing missing is a floor. An even, horizontal floor can be created with the `dCreatePlane()` function. The plane object is single-sided. Therefore collisions are only being registered on the side, which the normal vector of the plane is facing.

The `drawstuff` library is used to visualize the simulation. It comes included with the ODE library and provides basic functions to visualize a 3D scene. Fig. 19 shows the simulation in action.

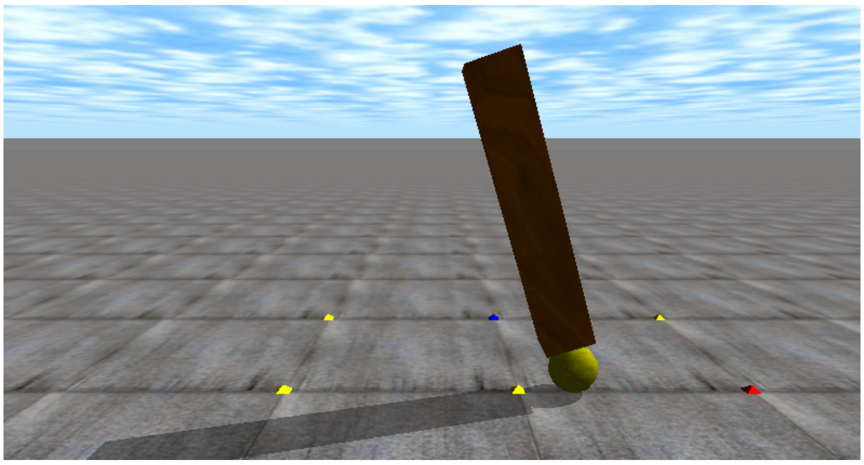

Figure 19: The ODE simulation is visualized with the `drawstuff` library

4.12.2. At run time

All the preparation work is done - now it is simulation time. Beforehand however a general short note: Setting directly positions or velocities for bodies should only be done to create the initial scene before the simulation. Once the simulation has been started, active bodies should be manipulated by applying force or torque to them. Otherwise discontinuities are introduced, which cause incorrect simulation results.

One time step in the simulation is carried out as follows:

- **Body collision detection**
  Collisions are modeled in ODE by creating temporary contact joints in the points where the bodies collide/intersect. All joints of this type are added to one `dJointGroupID` object. This way it is easier to discard them later.

  Two kind of collisions are assumed to occur in this particular simulation: The first kind is a collision between the wheel and the floor, which is expected for every time step. The second kind only occurs in case of a fall, where the upper body collides with the floor.

The collision surface properties are defined as follows: the static friction mu is set to `dInfinite`, the dynamic friction rho is set to zero and bounce as well is set to zero. This creates a "rolling without slip" condition for the wheel body on the floor plane.

- **Compute results for the next time step**
  This is done by calling `dWorldStep()`. ODE provides a second method, named `dWorldQuickStep(...)` which takes less computation time but is less precise.

- **Remove temporary collision joints**
  The function call `dJointGroupEmpty(...)` is used remove all collision joints, they have to be recalculated for the new body positions.

- **Update visuals**
  The two bodies are redrawn in their new pose with the drawstuff function calls `dsDrawBox(...)` and `dsDrawSphere(...)` respectively.

- **Update space state vector x**
  Listing 3 shows the corresponding code, the state space vector is stored as a Eigen library matrix object[4]. The wheel angle θ and lean angle ϕ are calculated using the `dBodyGetPosition(...)` and `dBodyGetRotation(...)` function. The `atan2(...)` function returns the lean angle with the correct sign. The angular velocities of the wheel $\dot{\theta}$ and the body $\dot{\phi}$ around Y-axis can be directly retrieved with the `dBodyGetAngularVel(...)` function.

```
Eigen::MatrixXd X(4,1);
const dReal* R = dBodyGetRotation(upperBody);
// R is 3x4 matrix, flattened, every 4. val empty
X(1,0) = atan2(R[2],R[10]); // phi
X(0,0) = -dBodyGetPosition(wheel)[0] / r_w + X(1,0); //
    theta
```

[4]The Eigen library is used in the written code whenever matrices or vectors are required. It is an open source C++ library and can be found under [36]

```
X(3,0) = dBodyGetAngularVel(upperBody)[1]; // dphi
X(2,0) = -dBodyGetAngularVel(wheel)[1] + X(3,0); // dtheta
```

Listing 3: Update of the state vector in the ODE simulation

4.12.3. Results

The system responses are generated for the same scenarios as in previous section. Again, the response to an initial 0.1 deg lean angle and an input step are being looked at. The non-linear system response (see section 4.11) is assumed as ground truth and compared to the results obtained with the simulation in ODE.

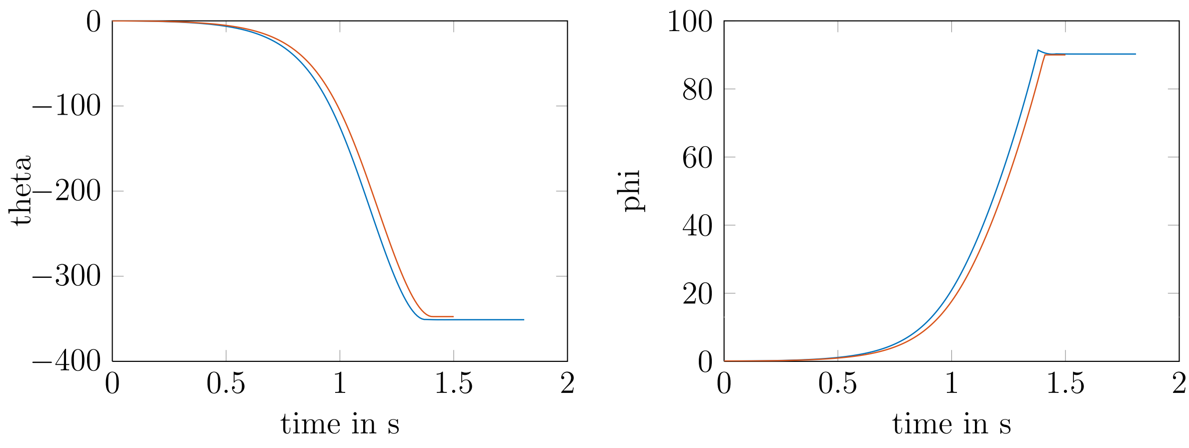

Figure 20: Fall from $\phi = 0.1$ deg lean angle, comparison between the non-linear system (red) and the ODE simulation (blue) result

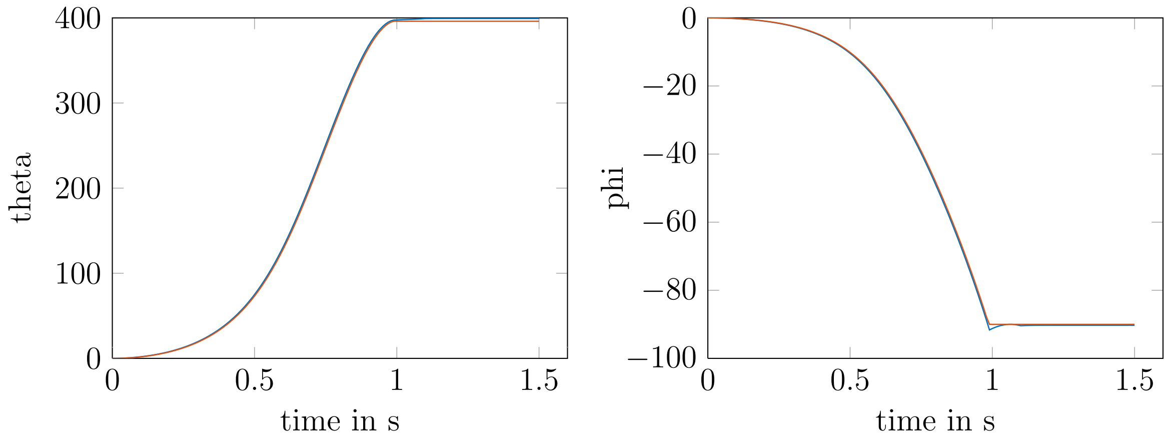

Figure 21: Input step at $t = 0$, comparison between the non-linear system (red) and the ODE simulation (blue) result

As can be seen Fig. 20 and Fig. 21, the response characteristics are overall similar. However, for the initial non-zero lean angle scenario the results are further apart.

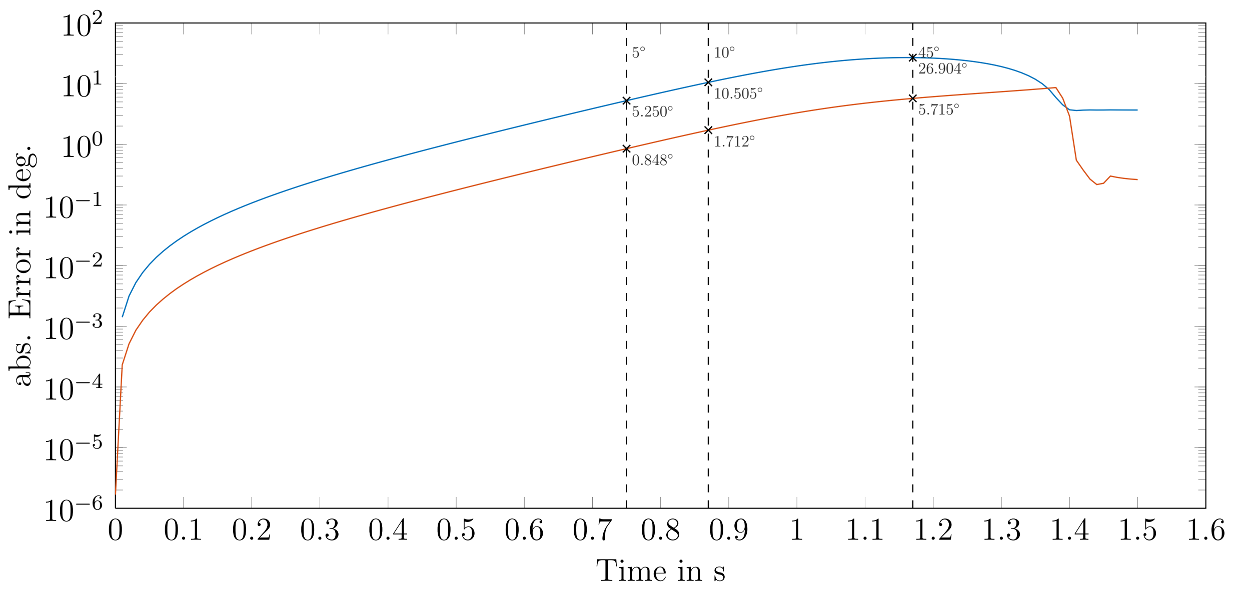

Figure 22: Error (logarithmic scaling on Y-axis)

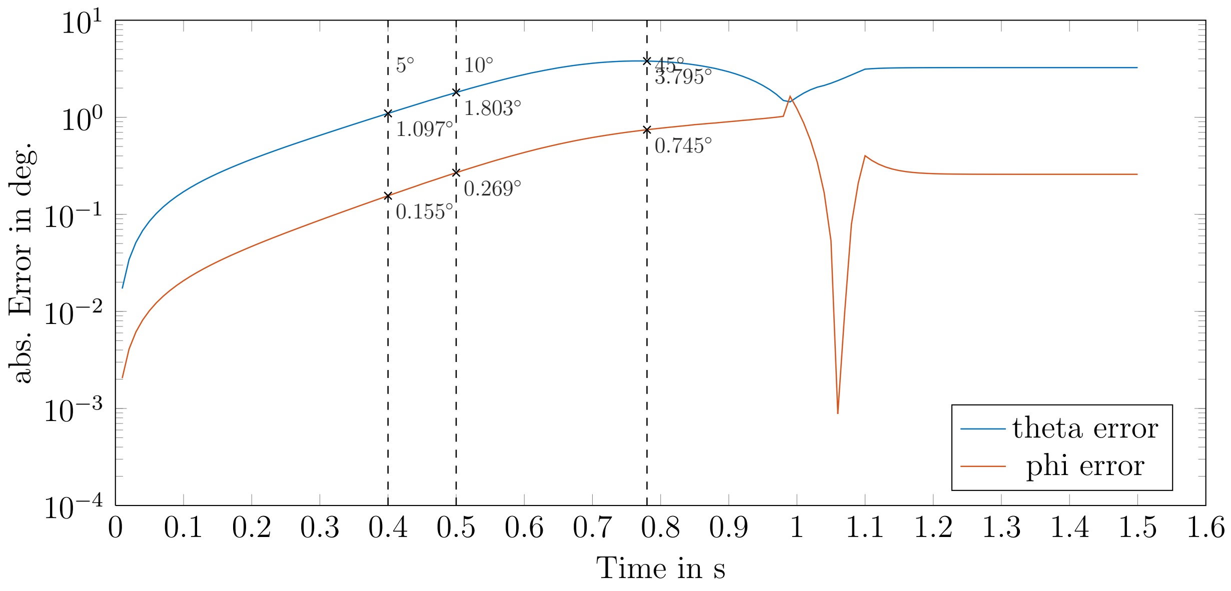

Figure 23: Error (logarithmic scaling on Y-axis)

A small spike can be seen for the lean angle in both scenarios. The lean angle shortly rises above 90 deg, which means the main body sinks into the floor plane due to an insufficient collision approximation and does not satisfy the contact constraint.

When the approximative solution in one time step does not satisfy a joint constraint, then a corrective force is applied in the next time step. The strength of the force can be adjusted with the Error Reduction Parameter (ERP). The second parameter provided by ODE to influence the joint constraint solving is named Constraint Force Mixing (CFM). An increased CFM value softens a rigid joint constraint. The ERP and CFM parameters cause a joint to behave like a spring damper system.

The ERP and CFM parameters can be set for each joint individually. Global default values can be set with the `dWorldSetCFM()` and `dWorldSetERP()` functions. For the simulation $CFM = 1e-16$ and $ERP = 0.3$ are used for all joints to keep them rigid and to achieve an error reduction with minimum overshoot.

5. Control

In the previous section an equation system describing the motion of the robot has been derived. Now in this section, based on the equation system, the design of an appropriate controller for dynamic stability - balancing - of the planar ballbot system is discussed. For balancing, the target of the controller is to maintain a zero lean angle, which means to keep the robot in an upright position by commanding an appropriate torque to the drive system of the robot.

A simulation in a physics engine has been introduced, which can be useful to test a control solution for the 2D planar model in 3D-space, once the controller design results for the planar ballbot system have proven successful.

5.1. State Feedback

This section describes how to implement a basic state feedback controller for the linearized system in state space. The state feedback in the state space is one way to modify the behavior of a system. The term "full state feedback" is used, when all variables of the state vector are used in the feedback path.

Choosing the right gain matrix K is the key to achieve the desired system behavior. The state vector x is multiplied by the gain matrix K in the feedback path. The difference between $K \cdot x$ and a reference input r is fed back into the system as input u. Therefore the new full closed loop system matrix A_{cl} becomes [28], [10]:

$$\dot{x}(t) = Ax(t) + Bu(t) \quad \text{where:} \quad u(t) = r(t) - K \cdot x(t) \tag{5.1}$$

$$\dot{x}(t) = \underbrace{(A - BK)}_{A_{cl}} x(t) + Br(t) \tag{5.2}$$

5.1.1. Controllability

Before an optimal feedback gain matrix K is designed for the planar ballbot system, the controllability matrix M_C of the system has to be examined. The controllability matrix M_C depends on the system matrix A and the input matrix B. It is calculated as follows:

$$M_C = \begin{pmatrix} A^0B & A^1B & \dots & A^{n-1}B \end{pmatrix} \tag{5.3}$$

where n is the number of state variables. In case the number of outputs $= 1$, the resulting M_C matrix is square. The controllability criteria is fulfilled, if $rank(M_C) = n$. For $rank(M_C) = n$, beginning from an arbitrary initial state x_0 every possible state x_f can be achieved in finite time by applying a certain input u. This is not the case, if for example one state variable cannot be influenced by the input. Then only a sub-system of A is controllable [10].

Back to the planar system of the robot, the controllability criteria is fulfilled for Eq.(4.39). State space, as the resulting matrix has full rank ($rank(M_C) = n = 4$). Given that relieving information, focus can shift back to the state feedback.

5.1.2. Linear Quadratic Controller

The gain matrix K directly influences the pole positions of the original system matrix A in the closed loop system. So a manual pole-placement method can be attempted. Alternatively the gains can be found by minimizing a quadratic cost function. The resulting controller is known as the Linear Quadratic Regulator (LQR) or the LQ controller, where linear stands for the linear system, and quadratic for the quadratic cost function. The quadratic cost function is defined as follows [28]:

$$\tilde{J} = \int_0^\infty \left(x(t)^T Q_x \, x(t) + u(t)^T Q_u \, u(t) \right) dt \tag{5.4}$$

The controlled system behavior can be tuned by adjusting the cost matrices Q_x and Q_u.

A common strategy is just to popularize the diagonal elements [28]. For this arrangement the cost matrix elements define how much each squared state variable and each squared input contributes to the total cost.

Returning from the short theory excursion to the planar ballbot system, the cost matrix Q_x and the scalar Q_u (only one system input exists) are now carefully selected: The lean angle ϕ and its angular velocity $\dot{\phi}$ should remain small, to make the system more predictable, less dynamic and to decrease the risk of a fall. Hence, a larger value for ϕ and $\dot{\phi}$ should be punished with an increased cost, whereas the wheel travel θ and the wheel velocity $\dot{\theta}$ are not considered to be as costly.

$$Q_x = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1000 \end{pmatrix} \qquad Q_u = 1 \tag{5.5}$$

The optimal gain matrix K for Q_x and Q_u is obtained in MATLAB® with the `lqr()` function. Fig. 24 shows the system response to a $\phi = 2$ deg initial lean angle and reference input $r = 0$. A higher value for Q_u reduces the commanded drive torque and increases the steady state time.

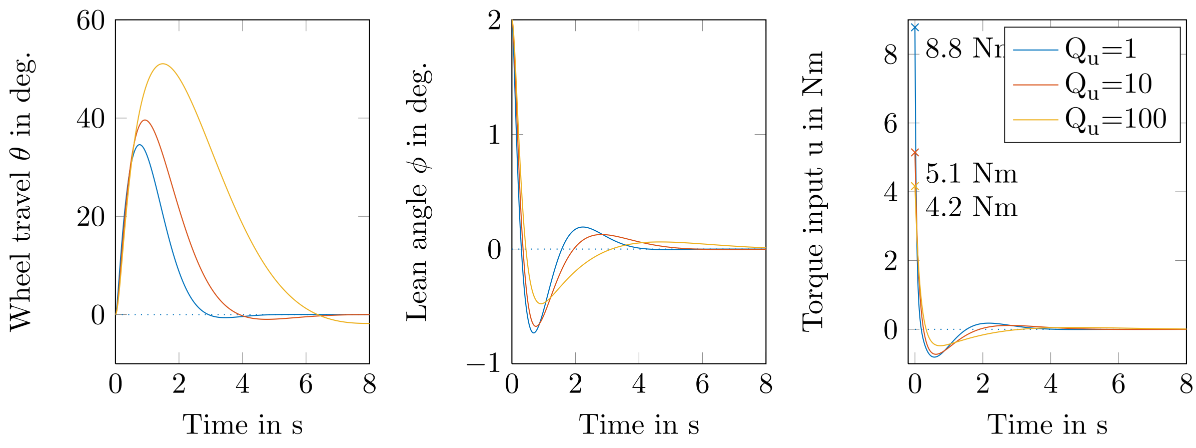

Figure 24: The influence of the input cost Q_u, with initial $\phi = 2$ lean angle

For a reference input $r = 0$, the augmented system attempts to reach zero for each state variable, which as well can be seen in Fig. 24. The initial non-zero lean angle ϕ is reduced by applying a torque to accelerate the wheel. Once the lean angle changes its sign, the robot returns to its origin, which gives all zero for the state variables.

For $r \neq 0$, the system behavior is yet not known and requires further examination. Therefore the augmented system is tested in the same scenario as before in section 4.11. where a step input is applied. So far this always resulted in a lean angle ϕ exceeding $\pm 90°$ - or put differently: The robot always took a hard landing on the floor.

The step response ($r = 1$) is displayed in Fig. 25. As can be seen, initially the wheel moves slightly backwards, which results in the robot starting to lean forward. Next, the wheel is accelerated in direction of the now increasing angular lean velocity of the upper body. This decreases the lean angle until it changes to the opposite. The wheel decelerates and the robot comes to halt at a different position. Hence, applying an input step sets a new position target. However, the reached steady state wheel angle θ does not equal the size of the input step.

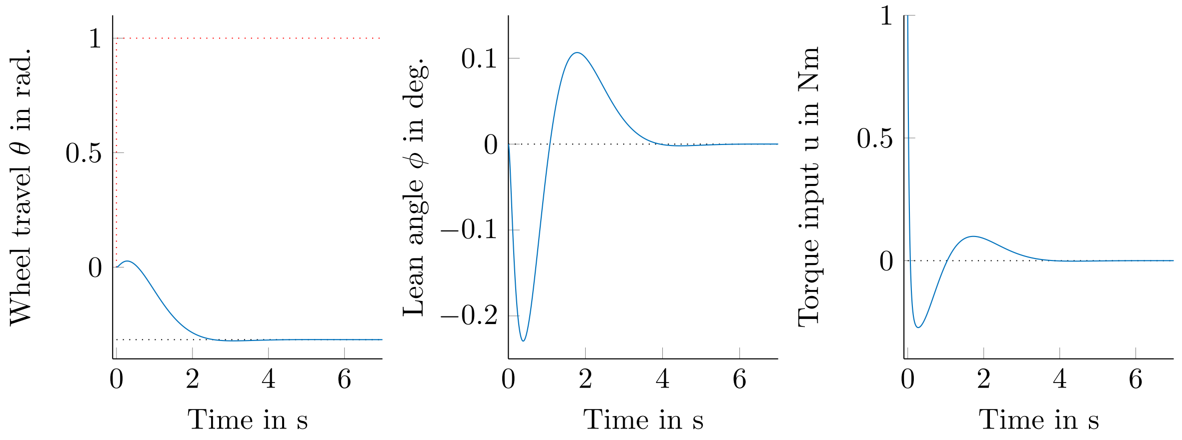

Figure 25: System step response

This issue is addressed by adding a precompensation to the reference input.

$$\dot{x}(t) = A_{cl}x(t) + B\tilde{r}(t) \quad \text{where:} \quad \tilde{r}(t) = V\,r(t) \tag{5.6}$$

$$\dot{x}(t) = A_{cl}x(t) + \underbrace{B\,V}_{B_v}\,r(t) \tag{5.7}$$

Assuming a steady state $\dot{x}(t = t_s) = 0$ at time t_s, the precompensation has to be chosen so that $y(t = t_s) - r(t = t_s) = 0$. For $D = 0$, the required precompensation can be calculated as follows:

$$V = -\left(C(A_{cl}{}^{-1}\,B)\right)^{-1} \tag{5.8}$$

With the added precompensation an unit input reference step results in a position change to $\theta = 1$ rad, see Fig. 26.

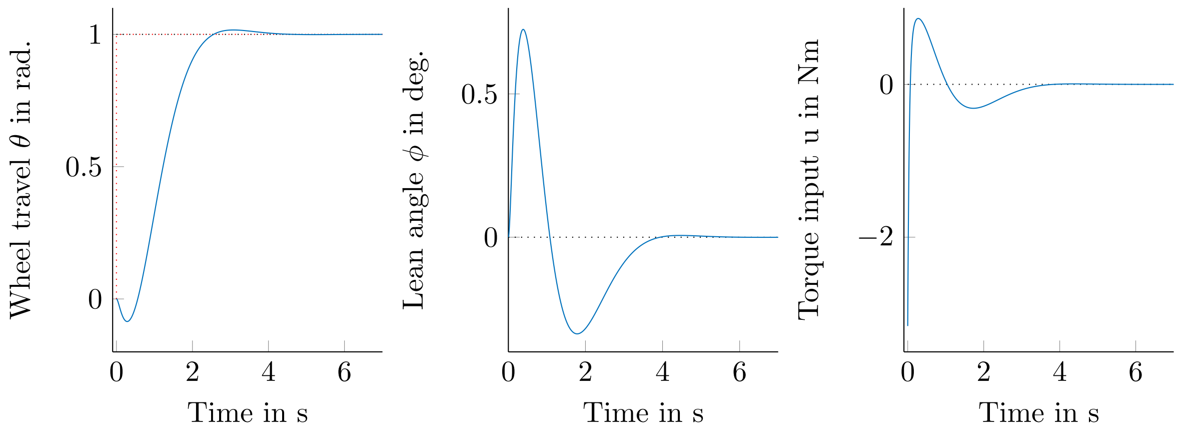

Figure 26: System step response with precompensation

5.1.3. Simulation

Two state feedback controllers are now added to the simulation in the open dynamics engine (see section 4.12), the first for the xz-plane, and a second one for the yz-plane. A sequential overlay image capturing the movement of the robot is displayed in Fig. 27. The robot starts at $x = y = 0$ m. The reference input is set to $r = 1/r_w$ for both controllers.

The same reference input is used in Fig. 28 and 29 . However, now between time

$t = 5$ s and $t = 27$ s disturbance forces are applied to the main body of the robot. This is accomplished in ODE with the `dBodyAddForceAtRelPos(...)` function. Despite the applied disturbance forces, the simulation remains stable. Once the disturbance forces are removed, the robot returns to its reference position ($x = y = 1$ m)

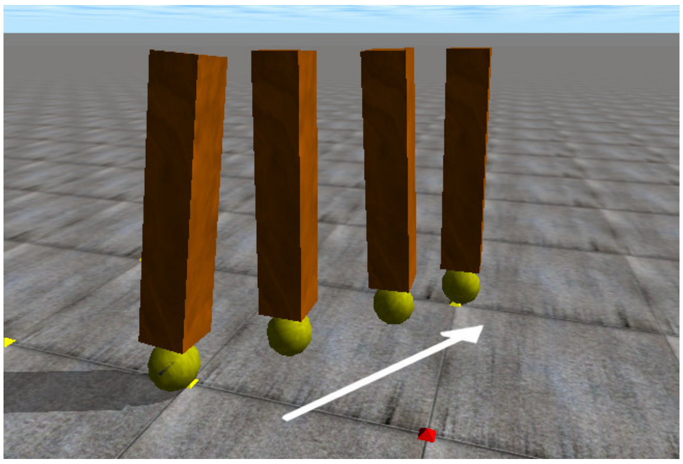

Figure 27: Two separate state feedback controllers stabilize the ballbot in the ODE simulation

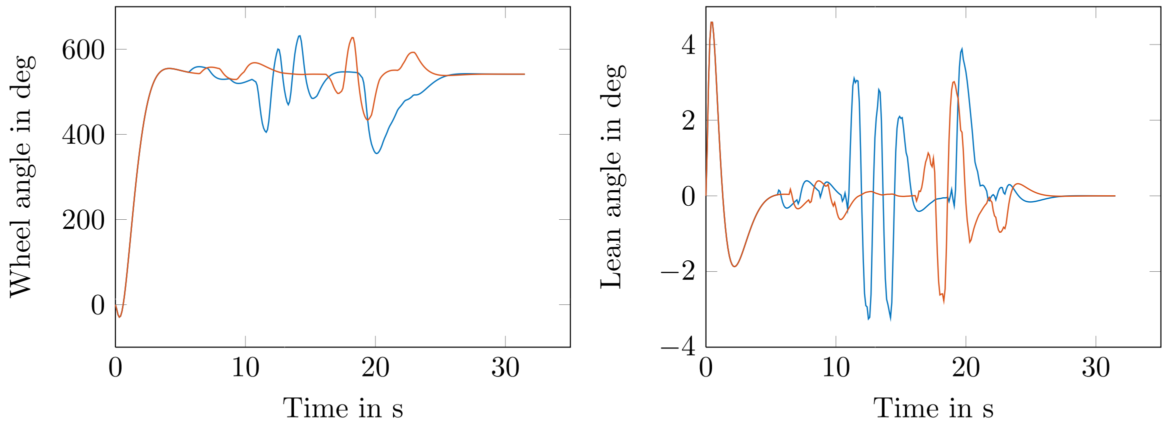

Figure 28: Two separate state feedback controllers, lean angle and wheel angle plot, with applied disturbance forces

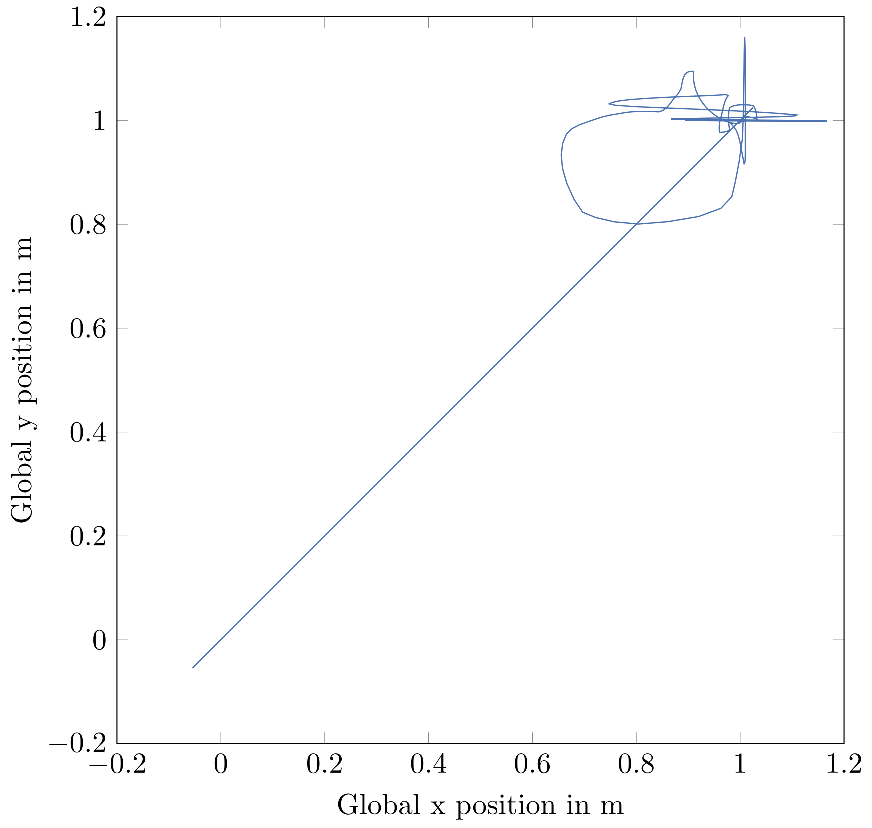

Figure 29: Wheel global position plot, with applied disturbance forces

5.2. Conventional Controller

In the current software implementation of the original ballbot a more conventional control structure is used [38]. It consists of an inner and outer closed loop. The inner loop is realized as a PID controller, which tracks the desired center of mass (COM) position x_d. This controller is sufficient for balancing and gives the robot an implied physical compliance. It can be moved around or pushed away with minimal force. The to be minimized error e_{com} is the difference between the desired COM position x_d and the actual COM position x_{com}:

$$e_{com}(t) = x_d(t) - x_{com}(t) \quad \text{where:} \quad x_{com}(t) = l_b \cdot \sin(\phi(t)) \tag{5.9}$$

The controller output u, based on the error e_{com}, is defined as

$$u(t) = k_p\, e_{com}(t) + k_i \int_0^t e_{com}(t)\, dt + k_d\, (\dot{x}_d(t) - r_w\, \dot{\theta}(t)), \tag{5.10}$$

where the global wheel velocity is approximated by the relative wheel velocity $\dot{\theta}$ observed from the upper body frame.

The outer loop control can be realized in different ways. For performing more complicated trajectories, an advanced planner software is used (see [33]) which feed-forwards lean angle and velocity values to the inner loop.An additional feedback loop corrects the commanded values based on the occurring error.

Solely for station keeping, the outer loop can be realized as a PD controller. It tracks a desired wheel position x_{wd}. The wheel position error e_w is defined as

$$e_w(t) = x_{wd}(t) - r_w(\theta(t) + \phi(t)) \tag{5.11}$$

This yields the following PD controller for station-keeping with lean angle output:

$$\phi(t) = k_p\, e_w(t) - k_d\, r_w\, \dot{\theta}(t), \tag{5.12}$$

under the assumption that $\dot{\phi}(t) \approx 0$ and the desired velocity $\dot{x}_{wd}(t) = 0$.

6. Implementation

The newly built robot is named SIMbot, as it is a ball**bot** actuated by a spherical induction motor (short: **SIM**). Fig. 30 shows a picture of the robot balancing on its own and gives an overview about the positioning of the most important components. In order to test

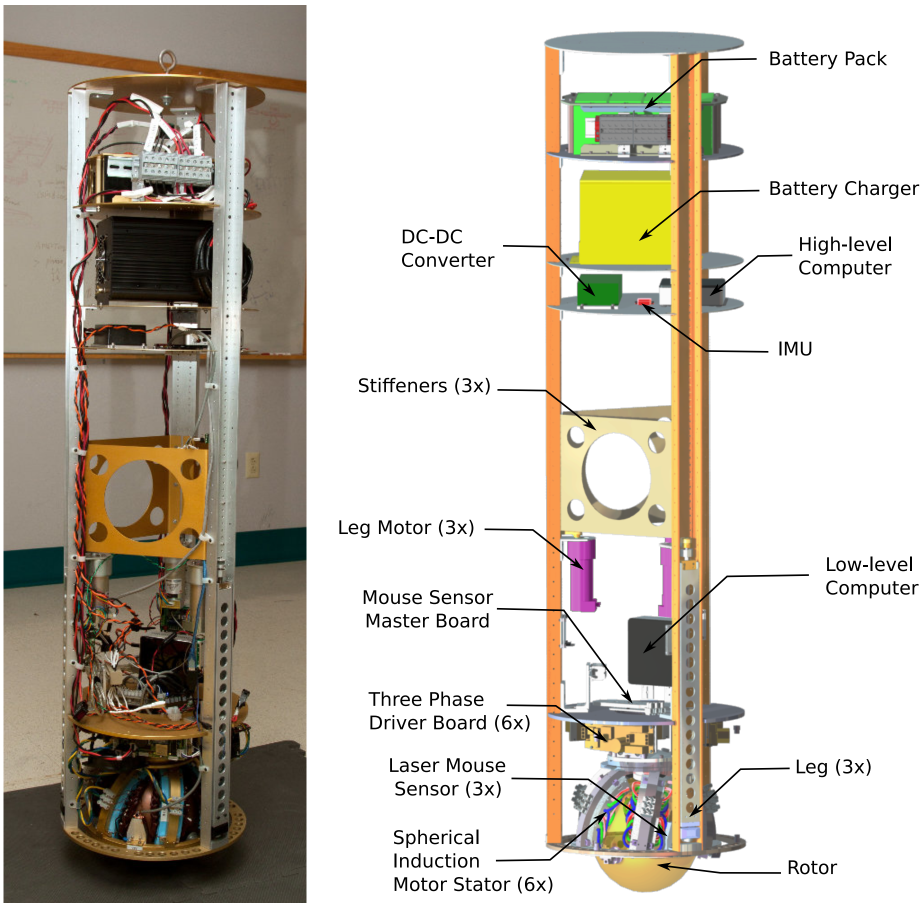

Figure 30: Finished SIMbot while balancing (left) and a CAD drawing (right), showing the positions of the most important components

the capabilities of the SIMbot, first a power distribution layout and a communication concept had to be designed and implemented. This was done having the concepts of the original ballbot as a guidance.

6.1. Power Distribution Layout

Fig. 31 displays the power distribution layout on SIMbot. Four non spillable 12 V lead acid batteries (4x Panasonic LC-R127R2P) connected in series supply power during mobile operation. In contrast to the original ballbot, the power distribution layout for SIMbot is capable of dealing with up to 96 V, just in case this would be needed for future tests or applications. SIMbot carries its own charger on board (Schauer JAC1548) to allow for easy recharging wherever. Each of the six Three Phase Driver Board is connected to the battery pack via the circuit breaker switch. The battery voltage is converted down to

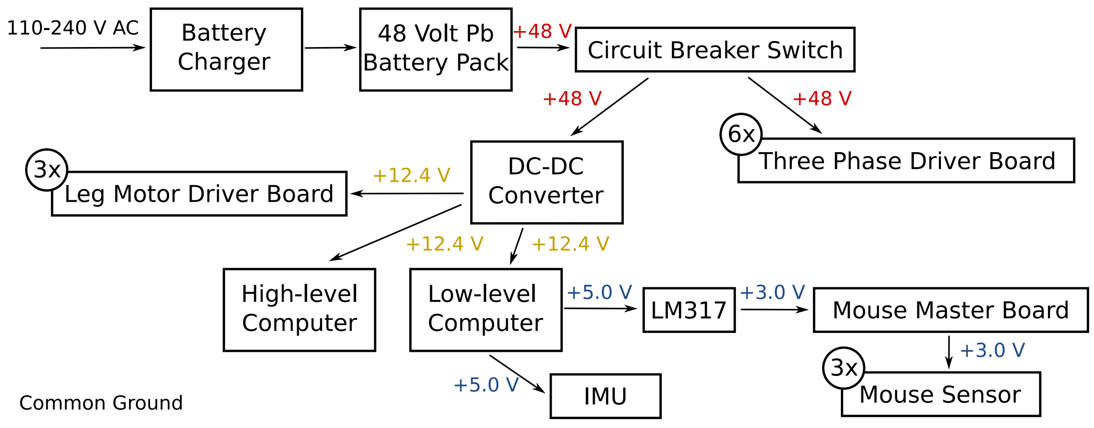

Figure 31: Power Distribution Layout

12.4 V via a DC-DC Converter (Delta Electronics B70SR12424A) capable of supplying a current of up to 24 A, which is more than sufficient to power both computers on board and the three leg motor drivers. The 5 V USB output voltage of the low-level computer is used to power the inertial measurement unit (IMU. and the Mouse Master Board and all three connected laser mouse sensors via an additional analog voltage regulator.

6.2. Communication and Processing

Fig. 32 focuses on the communication and the used protocols between subsystems. The robot carries two Intel architecture computers on board: The high-level computer, an

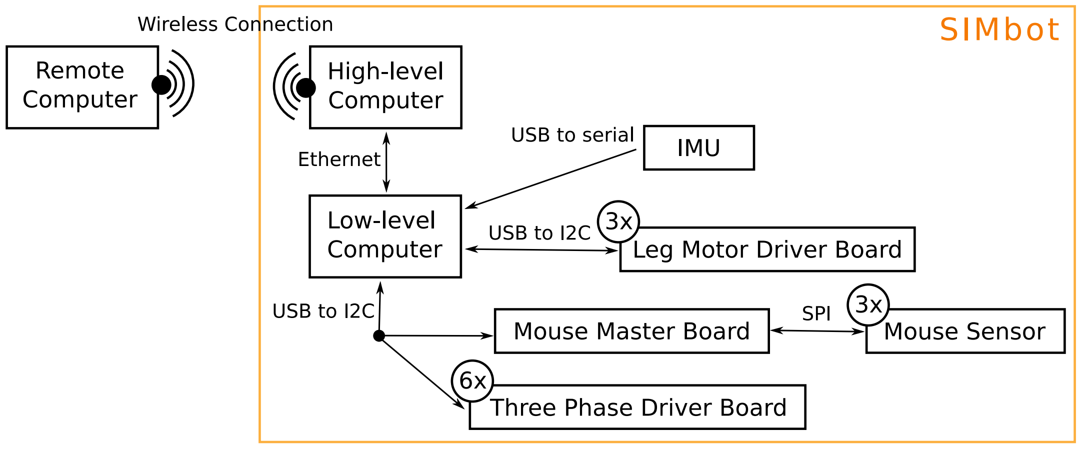

Figure 32: Communication protocols and paths on SIMbot

Intel NUC D34010WYK, is designated for future vision and path planning tasks.

The low-level computer used here is an Intel NUC 5i5MYHE. It runs a QNX real-time system and is used for the closed loop balancing control. The control loop is closed at 200 Hz. The low-level computer handles all communication with the six Three Phase Driver Boards of the spherical induction motor and reads sensor data from the Mouse Master Board and the inertial measurement unit (IMU).

The IMU sensor is a VectorNav VN-100. It supports sensor fusion and is configured to output Euler angles and angular velocities.

The Mouse Master Board deals with the low-level parts required for the communication with the three Avago ADNS-9800 laser mouse sensors and regularly retrieves data from the sensors.

Commands can be transmitted from remote via wireless to the high-level computer. These are consequently passed on to the low-level computer via a wired Ethernet connection.

7. Testing and Initial Results

7.1. Controller Setup

Two different control structures have been shortly introduced in section 5. For the implementation on the real robot, the conventional approach is selected. The main reason for this decision is the original ballbot, where the conventional control structure yields robust results. Furthermore, this enables SIMbot to make use of higher-level software, which already has been written for ballbot, like the trajectory planner.

Now, the first step is to find the right balancing controller gains. This is done experimentally. At the beginning only the proportional gain k_p is increased until SIMbot attempts to accelerate under larger lean angle errors. This gives a rough idea about the required order of magnitude for k_p.

Next the `BayesOpt` library is used, which implements the Bayesian optimization methodology for nonlinear-optimization, [24]. Given bounds for each gain and a cost function c, it attempts to optimize the gains for an unknown non-linear model. At first the bounds for the gains k_p, k_i and k_d are selected very conservatively to be on the safe side. Then, whenever the resulting gain set converges towards a gain bound, the bounds are extended for that particular gain and the optimization procedure is restarted.

The optimization procedure works as follows: First, the optimizer suggests a gain set. Five supervised short balancing trials, each with a duration of 5 sec, are carried out for the suggested gain set. The cost function c is evaluated for all of the five trials. The result is fed back into the optimizer. Based on the last and all previous results, the optimizer selects new gains and the whole procedure is repeated.

Compared to manual tuning, this procedure is systematic. However choosing the right cost function is crucial. At first, only the mean square error of ϕ_x and ϕ_y has been taken into account:

$$c = \frac{1}{n}\sum_{i=1}^{n}(0^\circ - \phi_x)^2 + \frac{1}{n}\sum_{i=1}^{n}(0^\circ - \phi_y)^2, \tag{7.1}$$

where ϕ_x and ϕ_y are the separate controller inputs, the lean of the SIMbot body in the xz- and yz- planes. This however leads to gain sets, where the spherical induction motor exhibits high frequency oscillations. To prevent this, the quadratic derivate of the lean angle error is additionally taken into account. Furthermore, the amount of commanded torque is another measure that can be taken into account. Using lower torque while keeping the lean angle error in the same range compared to some controller which commands higher torque should be rewarded by the cost function.

In the end, this systematic approach gives a good understanding about the usable range for each gain and their influence on the behavior.

7.2. On-board Mouse Sensor Adjustment

To determine a change of position of the robot over time, three laser mouse sensors are mounted on the frame of the spherical induction motor (see section 3.3.1). The resolution of the sensors is adjustable by writing a 2 byte value to the register over the SPI communication bus.

The default resolution register value rv is used (`= 0x44`). According to the sensor documentation, this gives approximately the following resolution [13]:

$$CPI_{ref} \approx rv \cdot 50. \tag{7.2}$$

The sensors return surface path length data as a number of "Counts" registered on their local x- and y-axis. With the CPI value (Counts per Inch) the Counts can be converted to the inch unit.

A linear error is assumed individually for each sensor and each x/y-axis, which should be compensated with a corrected CPI value. It is not necessary to compensate an offset, as all three sensors give zero-readings, when the robot is resting motionless.

The x-axis of each sensor should give the same reading, when a rotation of the rotor around the global z-axis is introduced. Correcting them is straight-forward. To determine the error for the y-axis of each sensor, the robot is moved multiple times a known distance forwards and backwards.

To ensure that the distance is always the same, stoppers are placed on the floor. The stoppers are angular aluminum brackets. So the robot is moved until its wheel hits one of the stoppers and then it is pushed back in the opposite direction until the opposing stopper is reached. The robot is hinged up in a movable frame, in order to prevent a lean angle influencing this procedure, see Fig. 33.

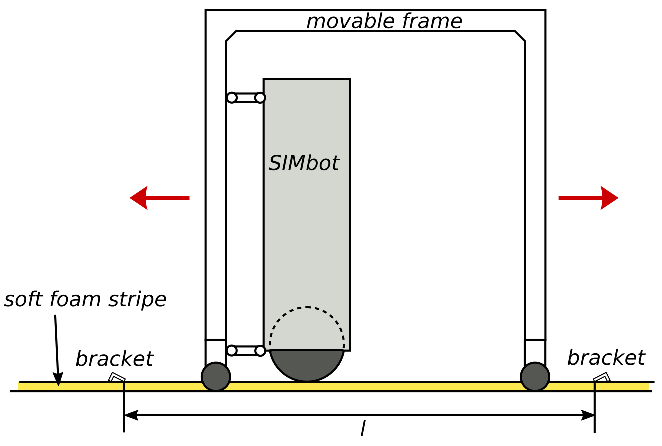

Figure 33: Sketch of test set-up to perform on-board mouse sensor adjustments

The distance l between the angular brackets has to be measured. This however is not equal to the required wheel travel distance. As shown in Fig. 34 a distance l_x remains between the center of the rotor and the edge of the bracket. Consequently, the reference distance l_{ref} is,

$$l_{ref} = l - 2 \cdot l_x \tag{7.3}$$

Measuring l_x directly is difficult as it is hard to determine the actual contact point between the wheel and the floor. However, measuring twice the distance, $2 \cdot l_x$, utilizing a second bracket, placed as shown in Fig. 34 opposing and parallel to the first one can

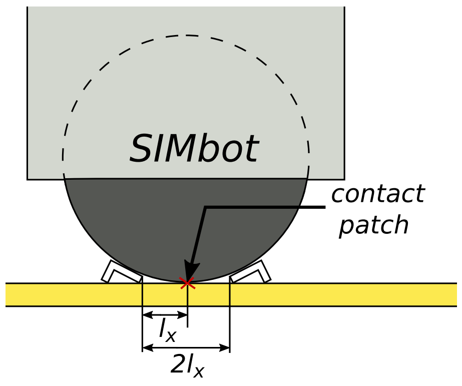

Figure 34: Definition of the length l_x

be done instead.

Three series of runs are performed. For each of them, the yaw of the robot is adjusted to have a different mouse sensor recording the main movement direction. The three corresponding direction unit vectors $v_{d0,1,2}$ are [5]:

$$v_{di} = \begin{pmatrix} \cos(120° \cdot i) \\ -\sin(120° \cdot i) \\ 0 \end{pmatrix} \qquad \text{for } i = 0, 1, 2 \tag{7.4}$$

Given Eq.(7.4) , the individual angular velocity unit vectors of the rotor are expected to be:

$$\omega_i = \begin{pmatrix} \sin(120° \cdot i) \\ \cos(120° \cdot i) \\ 0 \end{pmatrix} \qquad \text{for } i = 0, 1, 2 \tag{7.5}$$

Fig. 35 shows an exemplary run, v_{d2} is the robots main movement direction. This means, the sensor M_2 is facing in the main movement direction. The odometry results are shown in the Fig. 35 on the left side. The traversed path of the robot is colored according to time.

[5] equal to p_{mi}/r_w with $n_m = 3$ from Eq.(B.15)

First the main movement direction is computed from the data. Assuming that the main movement axis gives maximum variance, the principal components analysis is applied. Next the data dimension is reduced to one by projecting all x/y position coordinates onto the main axis. Subsequently the one-dimensional data is searched for local minima and maxima, assuming one zero-crossing in-between neighboring extreme values (Fig. 35 on the right side) . Taking the absolute differences between all found extreme values yields the measured distances along the main axis [15].

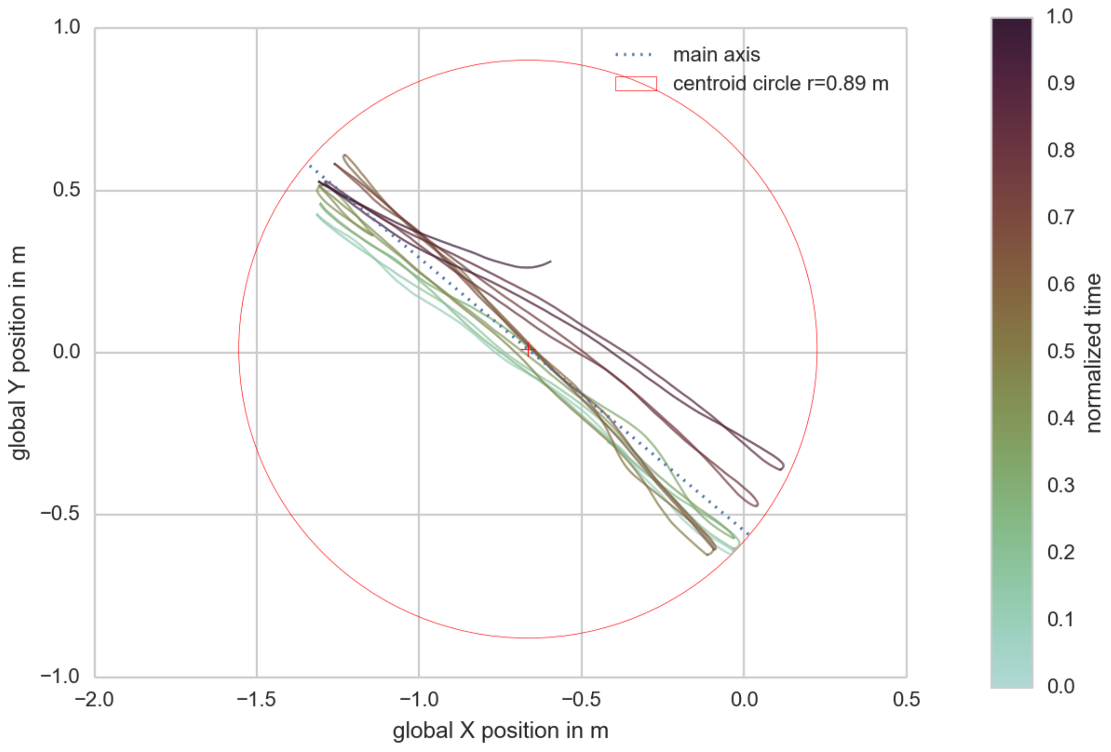

Figure 35: Exemplary odometry test result

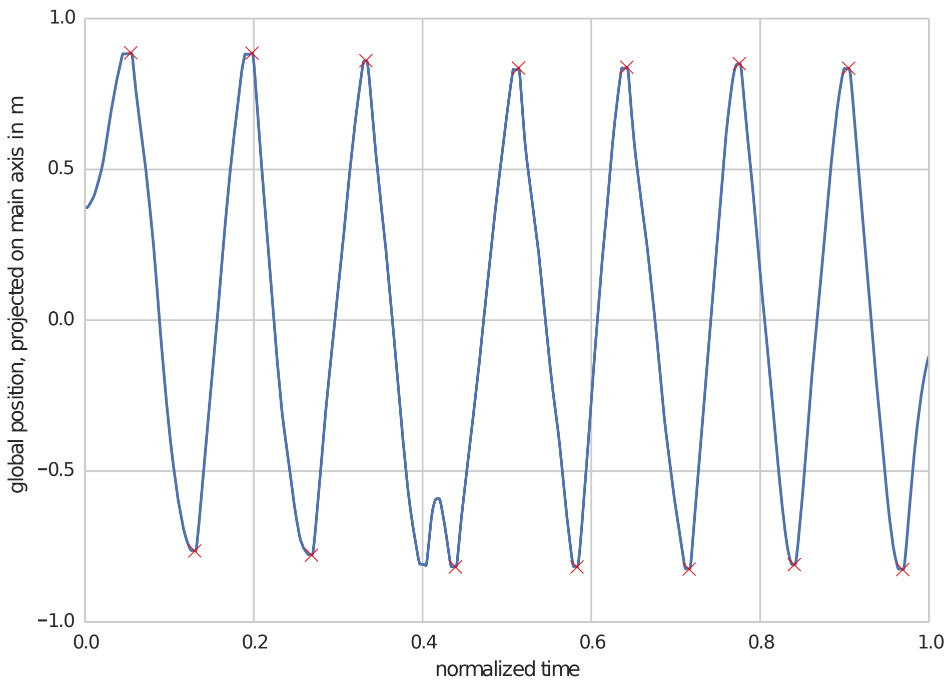

Figure 36: The data from Fig. 35, now projected on the main movement axis and reduced to 1D

Going back to section 3, Eq.(B.21) yields the local surface velocities given an angular velocity vector. Using the angular velocity vectors from Eq.(7.5) as input for Eq.(B.21) and applying the L1-norm to the results gives the individual sensor influences concerning the y-axis for each of the three measurement series:

$$s_0 = \begin{pmatrix} 1/2 \\ 1/4 \\ 1/4 \end{pmatrix} \qquad s_1 = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/2 \end{pmatrix} \qquad s_2 = \begin{pmatrix} 1/4 \\ 1/2 \\ 1/4 \end{pmatrix} \tag{7.6}$$

Given the information provided by Eq.(7.6), and the average distance result from each measurement series, the individual distances measured by each sensor can be retrieved:

$$\underbrace{\begin{pmatrix}(s_0)^T\\(s_1)^T\\(s_2)^T\end{pmatrix}}_{S_{sens}} \cdot \underbrace{\begin{pmatrix}l_{sens.0}\\l_{sens.1}\\l_{sens.2}\end{pmatrix}}_{L_{sens}} = \underbrace{\begin{pmatrix}l_{avg.0}\\l_{avg.1}\\l_{avg.2}\end{pmatrix}}_{L_{avg}} \tag{7.7}$$

$$L_{sens} = S_{sens}^{-1} \cdot L_{avg} \tag{7.8}$$

Finally, using Eq.(7.9) an adjusted CPI value can be calculated for each sensors y-axis:

$$CPI_{y.i} = \frac{CPI_{ref}}{l_{ref}} \cdot l_{sens.i} \qquad \text{for } i = 0, 1, 2. \tag{7.9}$$

Table 1 summarizes the adjustment results:

Table 1: Sensor adjustment results

$l_x =$	40.45 cm
$l_{ref} =$	1.656 m
$CPI_{ref} =$	3400 cnt/in

Series	Avg.	Std.
0	1.686 m	±0.52 cm
1	1.722 m	±0.81 cm
2	1.658 m	±1.06 cm

Sensor	$l_{sens.i}$	$CPI_{y.i}$
M_0	1.678 m	3355.42 cnt/in
M_1	1.566 m	3595.40 cnt/in
M_2	1.822 m	3090.23 cnt/in

7.3. Balancing

Only the two inner loop PID controllers are active for balancing. The following figures show an 8 s time extract of data recorded while balancing. The data was obtained on a floor consisting of soft foam tiles. This leads to additional damping, and the risk of damaging the copper surface of the rotor is minimized. The gains for both PID controllers are set to $k_p = 750$, $k_d = 55$ and $k_i = 55$ respectively. This gain set yields moderately good lean angle tracking (Fig. 37) while as well giving a sufficient physical compliance.

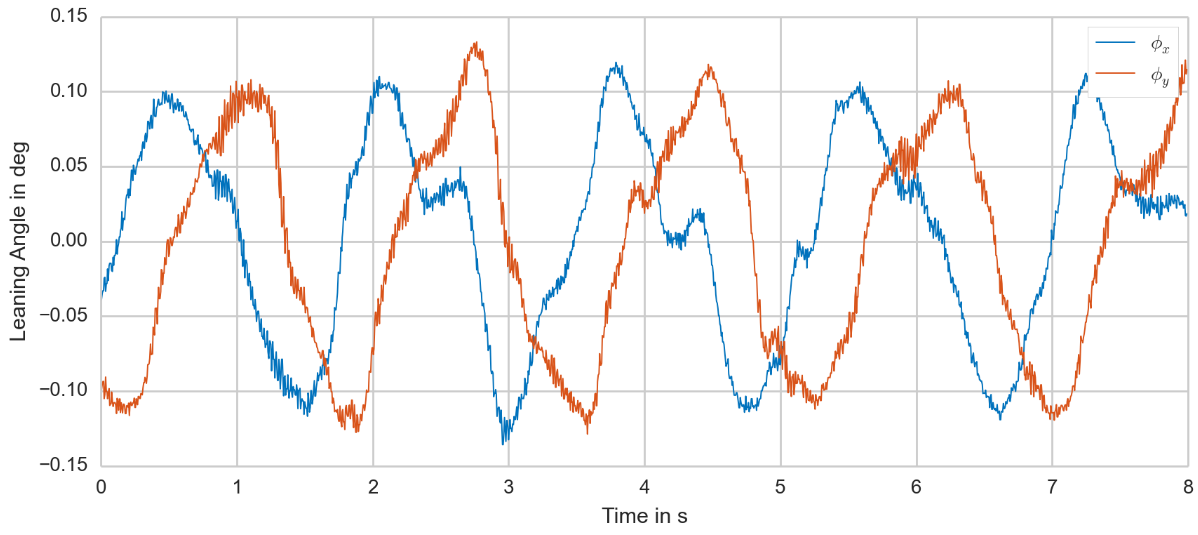

Figure 37: Lean angle vs. time

The lean angle tracking results can be seen in Fig. 37. The tracking target for balancing is a lean angle of 0 deg. The lean angle in x- and y-direction directly corresponds to the PID controller feedback values, whereas the actual body lean angle is at least as large as ϕ_x or ϕ_y. According to [37] the original CMU ballbot is able to maintain a lean angle of less than $\pm$ 0.1 deg. With the current controller implementation, SIMbot yields sightly more. The maximum observed lean angle is approx. $\pm$ 0.12 deg. A low lean angle tracking error is advantageous for executing pre-planned movement trajectories.

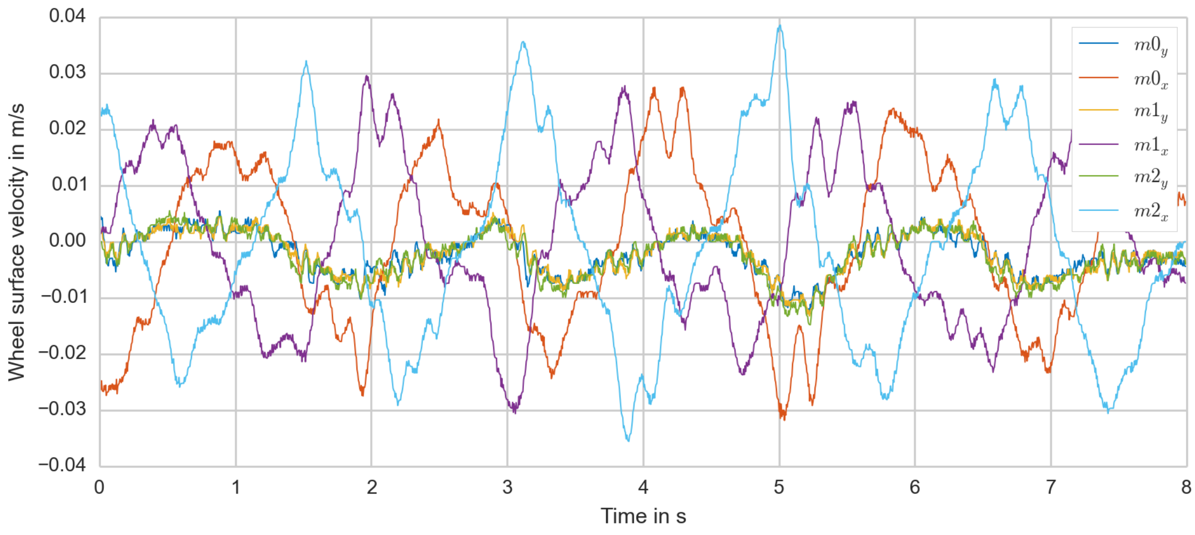

Figure 38: Wheel surface velocity vs. time

Fig. 38 depicts the surface velocities of the wheel. These are calculated based on the number of counts measured by the three laser mouse sensors.As to be expected, all three sensors give a similar reading on their y-axes. These correspond to the body yaw angle [6].

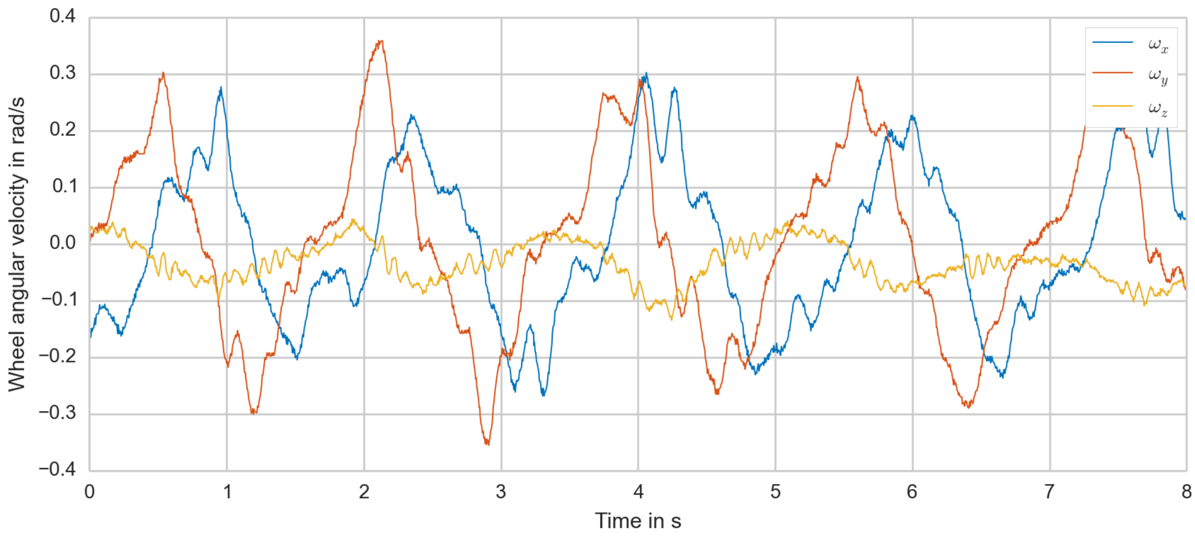

Figure 39: Angular velocity vector of the wheel vs. time

[6]assuming a non-slip condition between the wheel w of the robot and the floor

Converting the six measured surface velocities to an angular velocity vector, yields the plot depicted in Fig. 39.

Integrating the measured angular velocities ω_x and ω_y over time, plus compensating for the yaw and the actual lean angle gives a position plot in the global frame, as shown in Fig. 40. The robot performs a spiral-like movement, but stays within a patch with ≈ 7.5 cm radius.

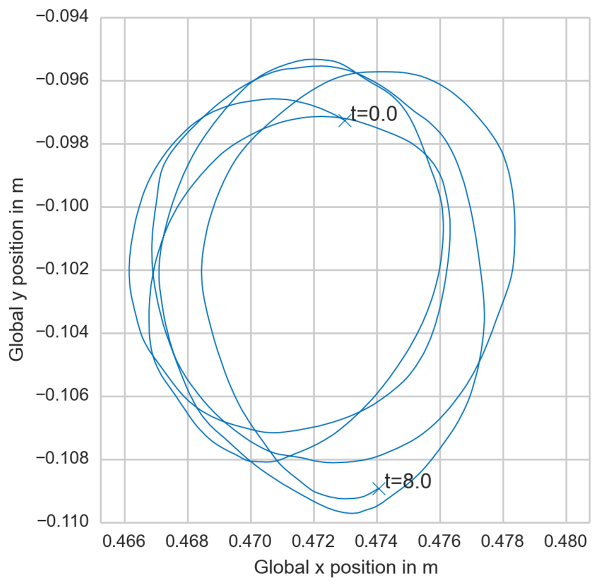

Figure 40: Global position plot of the ballbot

The commanded torque around x- and y-axis (τ_x and τ_y) can be seen in Fig. 41. There is no yaw angle control active, therefore the commanded torque $\tau_z = 0$. A maximum torque of $\approx \pm 5$ Nm (τ_x and τ_y) is demanded from the spherical induction motor. Going back to Fig. 39, it can be noted, that the maximum of the combined torque occurs at the same time as the maximum observed yaw velocity. This suggests, that the actual stator forces vary from their commanded strength, which consequently leads to an undesired non-zero τ_z, applied by the spherical induction motor to the main body of SIMbot.

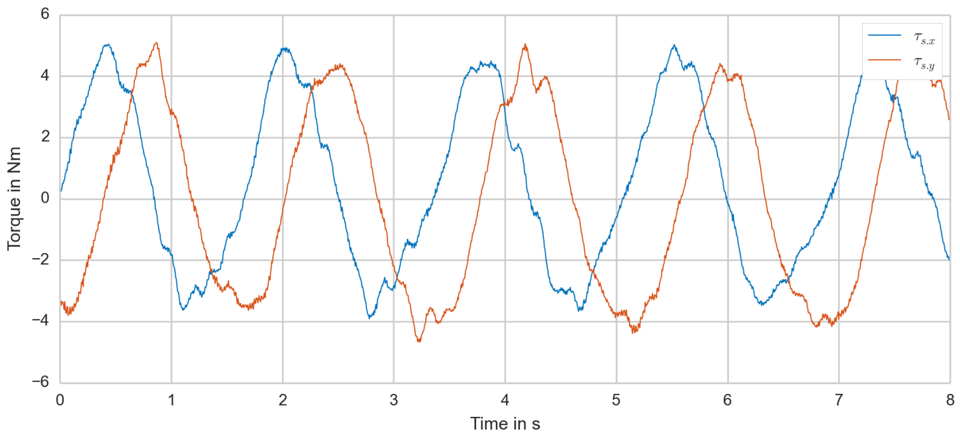

Figure 41: Torques τ_x and τ_y vs. time

7.4. Station Keeping

For station keeping, each of the two PID controllers is separately augmented with a station keeping controller. The station keeping controller consists of a proportional gain and a damping gain (PD). The gains were manually tuned and are set to $k_p = 3.0$ and $k_d = 0.7$. Fig. 42 displays a 48 sec time period, in which the robot receives multiple pushes to demonstrate the station keeping capabilities. Each push direction is indicated with an arrow. During this time period SIMbot always returns to its initial spot with an accuracy of $\approx \pm 3.6$ cm in x-direction and $\approx \pm 2$ cm in y-direction. Each push is applied for ≈ 2 s with a force of $8 - 10$ N. This force is applied manually and measured by a force sensor. The resulting largest displacement from its initial position is 38 cm.

Fig. 43, 44 and 45 show the behavior focusing on a single push. The push is initiated at $t = \approx 1.8$ sec. This can be seen in the plot on the right in Fig. 43 which displays the global x and y-coordinates projected onto the main movement axis.

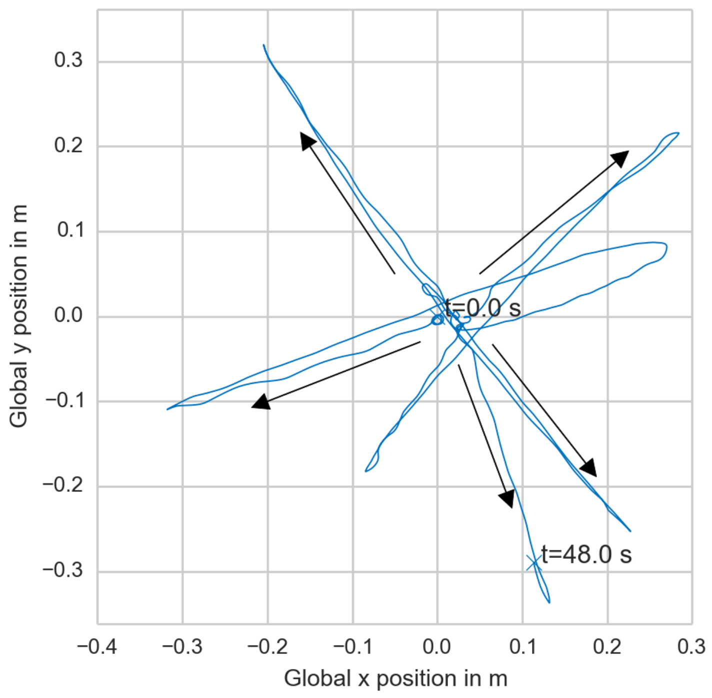

Figure 42: Global position station keeping plot

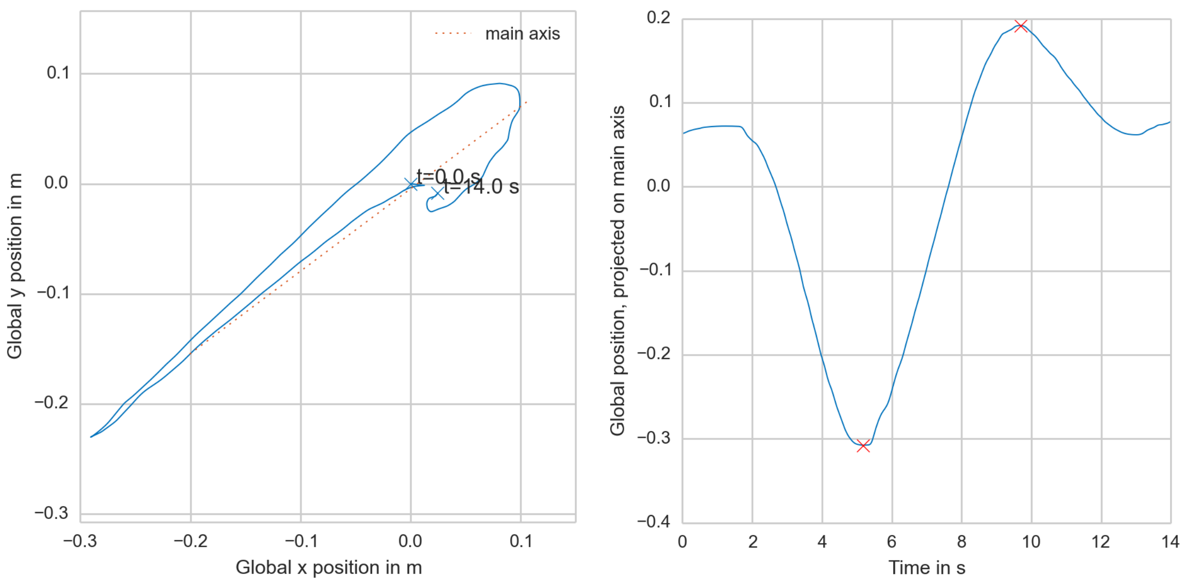

Figure 43: Global and projected position of SIMbot

The lean angle plot in Fig. 44 shows how the body of SIMbot starts leaning against the direction of the occurring push due to the active PD-controller. Beginning at $t = 7.5$ sec until $t = 12$ sec an overshoot occurs, which as well can be observed in the global position plot in Fig. 43.

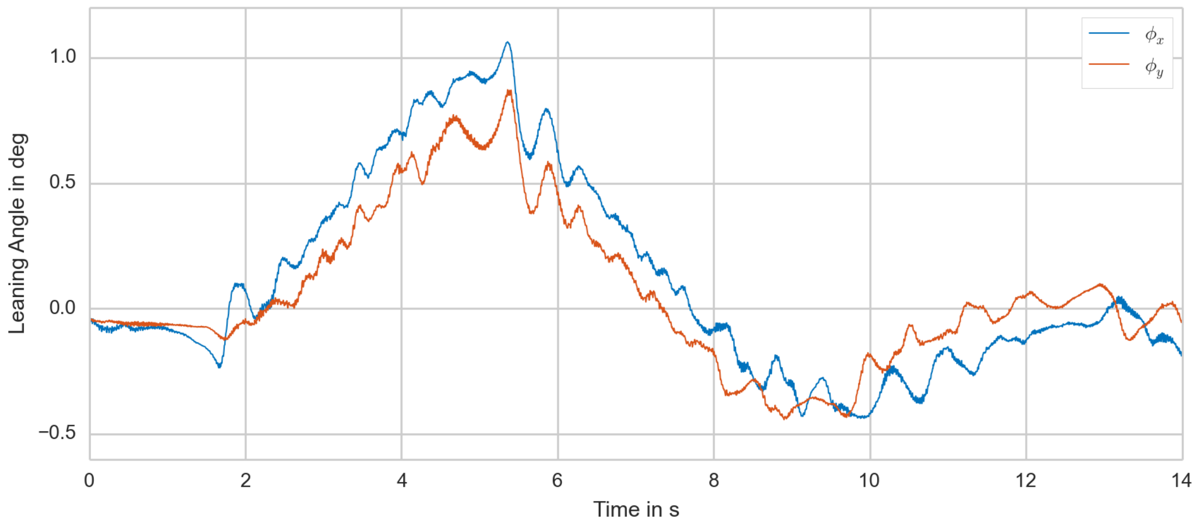

Figure 44: The lean angle ϕ_x and ϕ_y of SIMbot

The torque limit is set to 8 Nm for each component of the vector τ_s. Depending on the resulting torque vector, a stator force limit $F_{s.max}$ might be reached before hitting the maximum 8 Nm. Fig. 45 however indicates no capped torque output during the station keeping maneuver.

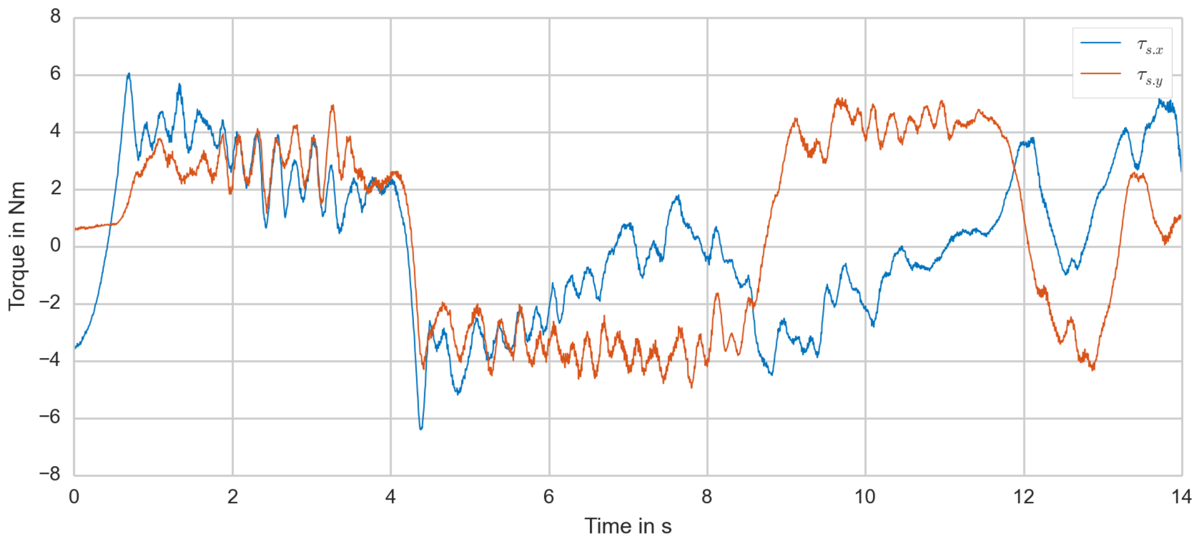

Figure 45: The commanded torque vector components $\tau_{s.x}$ and $\tau_{s.y}$

7.5. Trajectories

A simple point to point trajectory is demonstrated with SIMbot. A planning software which has been written for the original ballbot (see [34]) is reused for this task. The movement direction is aligned with the global x-direction. Therefore Fig 46, 47 and 48 only show one coordinate.

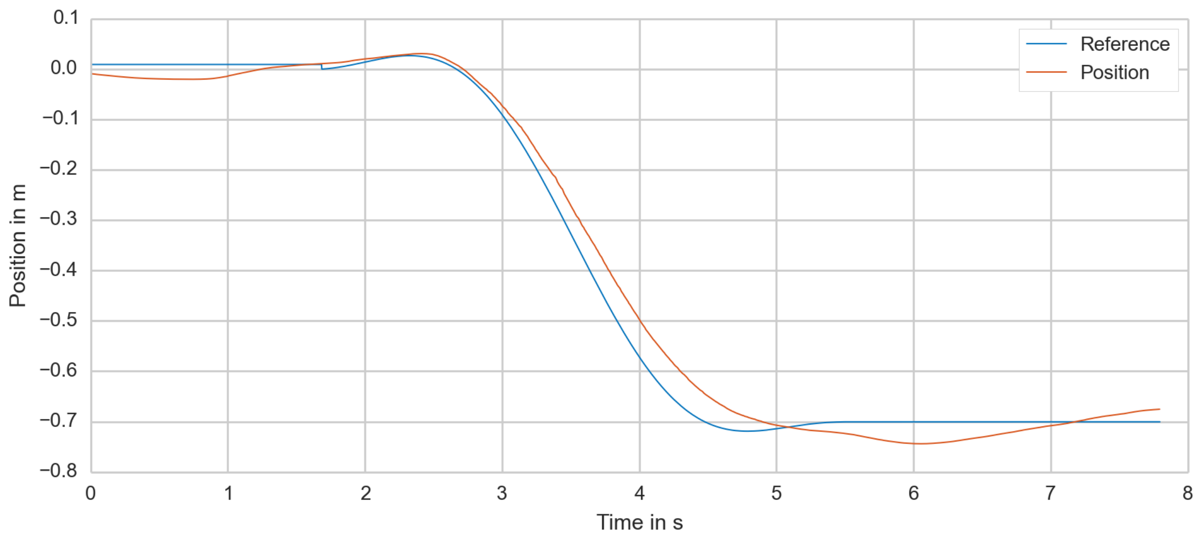

Figure 46: Position plot

SIMbot reaches the end of the 0.7 m long trajectory, however takes slightly longer than planned. The velocity plot in Fig. 47 reveals some uneven, or wobbly, angular velocity of the wheel. To a certain extent, this is as well reflected in the lean angle plot in Fig. 48. A possible explanation for this is a changing friction torque in the drive mechanism caused by the ball transfer units.

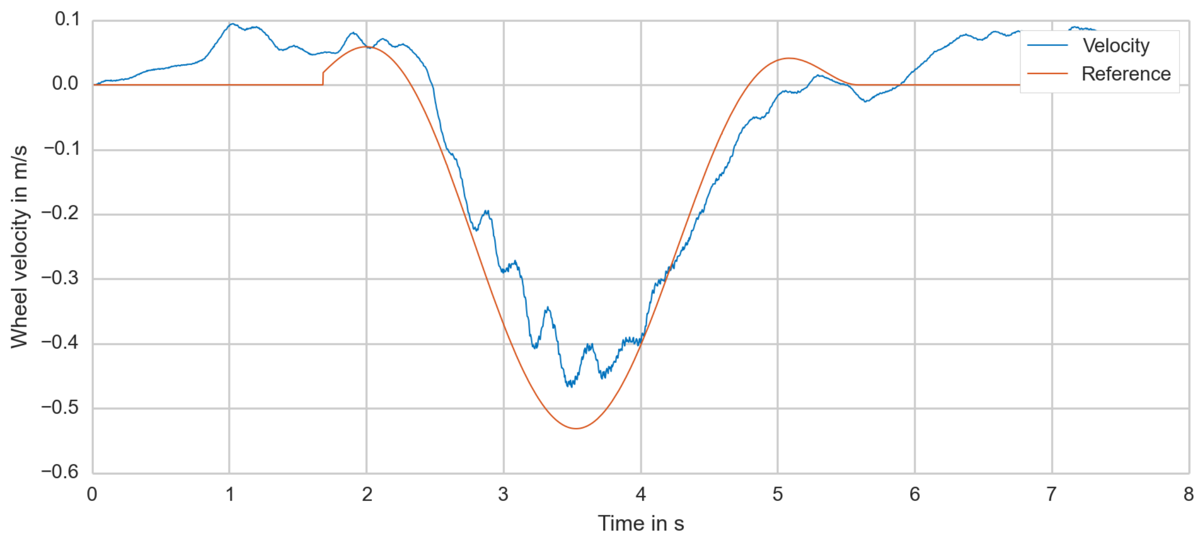

Figure 47: Velocity plot

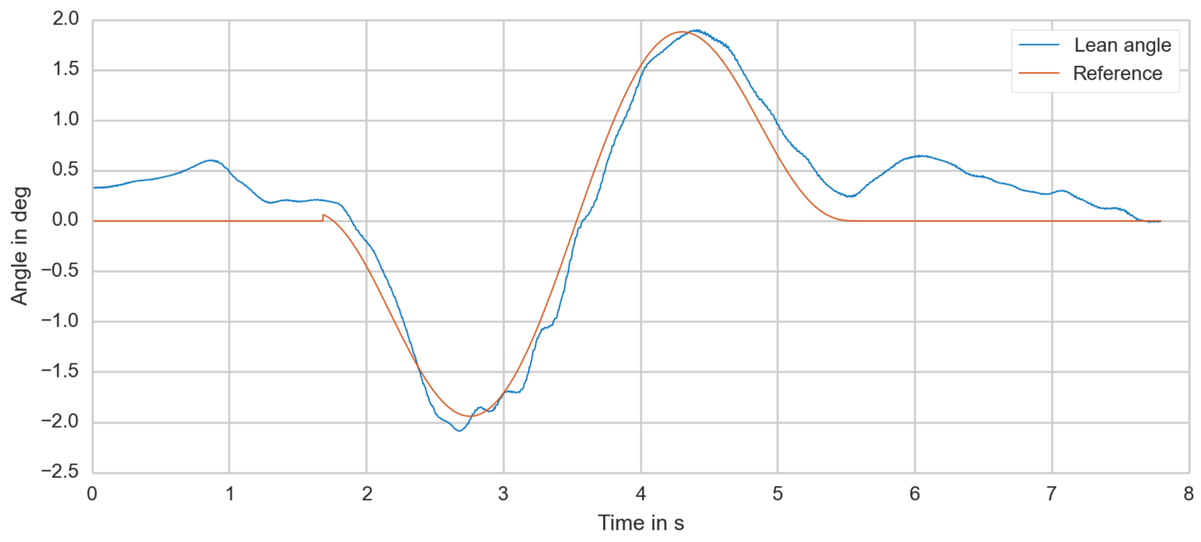

Figure 48: Lean angle plot

8. Conclusions

The initial attempts for stabilizing and driving a ballbot with the novel spherical induction motor were successful. The results show that the maximum generated torque and the dynamic characteristics, especially the rise time, of the current spherical induction motor are sufficient for balancing, station keeping and for traversing trajectories.

As well, the current design of the spherical induction motor is able to bear the added mass of the robot body without negative implications. Thanks to the advancements in electronics, the required computation power is realized in a much smaller package, compared to the original ballbot. This leaves more space for additional modules and future applications on the robot.

Comparing the obtained results to the drive mechanism of the original ballbot, it becomes clear that the spherical induction motor is yet no match for the mechanical drive system. This however is not a concern, as the main goal has been to figure out whether this novel technology can be applied to mobile balancing robots after all.

Now that this has been affirmed, improving the yielded torque output and efficiency of the drive system will be the next steps. For the results presented in this work, the capabilities of the spherical induction motor are limited by the maximum current that the driver boards can supply to the stator coils. There are also thermal problems to be solved. A new driver board with higher performance is in work, which will develop less heat and allow for more current.

The air gaps between the rotor and the stators can be further reduced to increase the efficiency of the drive. The current design does not allow for individual adjustment of the gap between the stator element and the spherical rotor. Instead each stator is mounted at a fixed distance on the stator support frame. To allow for effortless testing of different air gap sizes and to compensate for manufacturing tolerances of the stator support frame, it seems useful to have each stator individually adjustable in a future revision.

Once those improvements have been implemented, a more thorough comparison of both drive systems regarding their energy consumption and their dynamic capabilities can be attempted in mobile robots with same mass and inertia. .

Another area for future work is the bearing of the spherical wheel. In both, the ballbot and the SIMbot, three ball transfer units (with 1 inch ball diameter) carry the static and dynamic load of the robot body. The performance of the ball transfer units degrades due to dirt and other abrasive particles over time, see Fig. 49, left image. This causes the friction to become less homogeneous. Worst case dirt particles block the supporting balls internally and lock the position of the bearing, see Fig. 49, right image. These kinds of drastic abrupt friction changes make balancing a lot more unstable, or even might create a large enough disturbance to cause a fall. An advanced solution which would eliminate

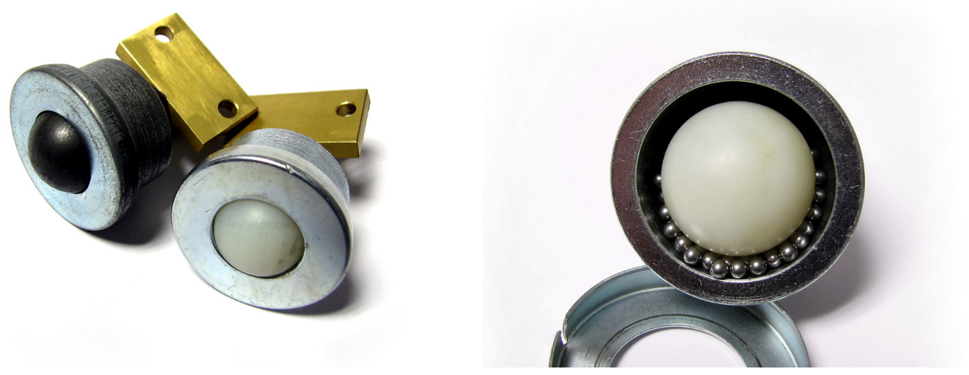

Figure 49: A used ball transfer unit next to a new one (left) - An opened ball transfer units with its supporting balls exposed (left)

these discussed problems is to replace the mechanical ball transfer units with air bearings. As a downside however an air pressure pump would need to be added to the robot.This as well will contribute to the overall power consumption, still the added benefits could easily out-weight the drawbacks.

Finally, measures have to be taken to enable SIMbot to traverse hard floor surfaces without damaging the outer copper layer of the spherical rotor. One idea is to add a thin resilient coating to the spherical rotor. More effective however could be configuration with a soft interposer ball, acting between the spherical rotor and the floor, as demonstrated on BallIP, [18].

A. Physical System Parameters

Table 2: Physical system parameters

Parameter name	**Symbol**	**Value**
Mass of the body	m_b	41.975 kg
Mass of the wheel	m_w	7.45 kg
Moment of inertia of the wheel	I_w	$0.05\ \mathrm{kg} \cdot \mathrm{m}^2$
Moment of inertia of the body [7]	I_b	$12.16\ \mathrm{kg} \cdot \mathrm{m}^2$
Wheel radius	r_w	$0.1013 \pm 0.00012\ m$
COM of body above center of wheel	l_b	0.679 m
polar angle of stators	θ	40 °
skew angle of stators	γ	10 °

B. Spherical Induction Motor

B.1. Torque Output

According to [3], each stator i is assumed to generate a force F_{si} in its volume center, projected onto the spherical rotor surface. Fig. 50 shows a $n_s = 6$ evenly spaced stator configuration. Each stator is represented in the figure as a blue arc.

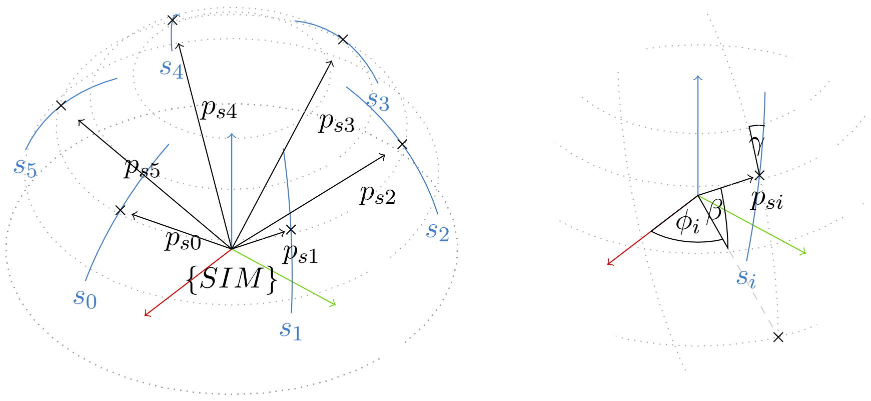

Figure 50: A six stator ($n_s = 6$) configuration (left) and the angles ϕ_i, β and γ for stator i (right)

A position vector p_{si} is defined for each stator. The vector p_{si} gives the position of the stator center on the sphere surface in the spherical induction motor coordinate frame. Hence, first a rotation matrix R_i is created according to Fig. 50.

$$R_i = \begin{pmatrix} c_{\phi_i} & -s_{\phi_i} & 0 \\ s_{\phi_i} & c_{\phi_i} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_\beta & 0 & -s_\beta \\ 0 & 1 & 0 \\ s_\beta & 0 & c_\beta \end{pmatrix} = \begin{pmatrix} c_\beta c_{\phi_i} & -s_{\phi_i} & -s_\beta c_{\phi_i} \\ c_\beta s_{\phi_i} & c_{\phi_i} & -s_\beta c_{\phi_i} \\ s_\beta & 0 & c_\beta \end{pmatrix} \tag{B.1}$$

The angle ϕ_i defines each stator location in the xy-plane, while β describes the elevation angle. Taking the first column of R_i and multiplying it by r_w (the sphere radius) yields the stator position vector p_{si}:

$$p_{si} = R_i \cdot \begin{pmatrix} r_w \\ 0 \\ 0 \end{pmatrix} = r_w \cdot \begin{pmatrix} c_\beta c_{\phi_i} \\ c_\beta s_{\phi_i} \\ s_\beta \end{pmatrix} \tag{B.2}$$

To allow for torque generation around the z-axis, all stators are slightly skewed by the skew angle γ. The rotation axis around which γ is applied equals the first column in the previously defined rotation matrix R_i.

Given this information each stator force vector F_{si} can be formulated:

$$F_{si} = R_i \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\gamma & s_\gamma \\ 0 & -s_\gamma & c_\gamma \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ |F_{si}| \end{pmatrix} = \begin{pmatrix} -s_{\phi_i} s_\gamma - s_\beta c_{\phi_i} c_\gamma \\ c_{\phi_i} s_\gamma - s_\beta c_{\phi_i} c_\gamma \\ c_\beta c_\gamma \end{pmatrix} |F_{si}| \tag{B.3}$$

In general a torque vector τ is the vector-crossproduct of r and force vector F, where r is a position vector, perpendicular to F, with its origin in the axis of rotation:

$$\tau = r \times F \tag{B.4}$$

In this particular application, r from Eq.(B.4) equals the defined stator position vector p_{si}. When each stator force $F_{si}$is given, the resulting torque τ_s of the spherical motor can be calculated by summing up all acting stator torque vectors:

$$\tau_s = \sum_{i=0\ldots n_s-1} p_{si} \times F_{si} \tag{B.5}$$

The acting direction of each stator torque τ_{si} always remains constant, only the torque strength changes according to $|F_{si}|$. Therefore Eq.(B.5) can be rewritten to

$$\tau_s = \cdot \sum_{i=0\ldots n_s-1} |F_{si}| \cdot \underbrace{(p_{si} \times f_{si})}_{t_{si}}, \tag{B.6}$$

where t_{si} is a vector, describing the torque introduced by an unitary force f_{si} of stator i on the rotor. In the next step, the sum is rewritten to a matrix notation:

$$\tau_s = \underbrace{\begin{pmatrix} t_{s0.x} & \cdots & t_{s(n_s-1).x} \\ t_{s0.y} & \cdots & t_{s(n_s-1).y} \\ t_{s0.z} & \cdots & t_{s(n_s-1).z} \end{pmatrix}}_{T} \cdot \underbrace{\begin{pmatrix} |F_{s0}| \\ : \\ |F_{s(n_s-1)}| \end{pmatrix}}_{F} \tag{B.7}$$

The matrix T in Eq.(B.7) does not depend on τ_s or F and therefore is constant, given the condition that the stator positions do not change physically.

Given a torque vector τ_s and searching a solution for the F vector, there exist more unknown force variables [8] than equations, which means the normal inverse of T cannot be used. However, by reversing Eq.(B.7) with the Moore-Penrose pseudo-inverse (left inverse, $T^+T = I$) [6], it is possible to calculate each stator force for a given torque vector τ_s:

$$\underbrace{\left((T^T T)^{-1}\, T^T\right)}_{=T^+\text{...pseudo-inverse}} \cdot \tau_s = \underbrace{\left((T^T T)^{-1}\, T^T\right) \cdot T}_{I}\, F \tag{B.8}$$

$$T^+ \cdot \tau_s = F \tag{B.9}$$

The pseudo inverse T^+ yields the least square optimal solution. The T^+ matrix does not depend on time-dependent variables, and therefore only needs to be calculated once and not for every time step.

B.1.1. Torque limit

The size of the torque vector τ_s is limited by the maximum thrust force $F_{si.max}$, that can be individually generated by each of the n_s stators. In a scenario where one component of the resulting stator force vector F exceeds the maximum, the following procedure can

[8] when the number of stators $n_s > 3$

be performed to reduce the torque vector adequately:

$$\tau_s = \begin{cases} \tau_s \cdot \frac{F_{si.max}}{max(abs(F))} & \text{if } max(abs(F)) > F_{si.max} \\ \tau_s = \tau_s & \text{otherwise} \end{cases} \tag{B.10}$$

The torque strength is reduced, whereas the direction of the torque vector remains unchanged. Just limiting the one exceeding stator force causes an uncontrolled new directional component in the resulting torque vector.

For the application of the spherical induction motor in the Ballbot, a higher torque is expected to becommanded about the x- and y-axis. Torque around the z-axis is not as important, because it is not required to stabilize the robot.

The force limiting approach, shown in Eq.(B.10) is acceptable, if keeping the direction has the highest priority. However, concerning the ballbot, applying the maximum available torque when needed for stabilization, should be considered a higher priority. A change of direction for the torque vector is acceptable, as long as the torque applied around the z-axis remains unchanged, to avoid a spinning of the upper body of the robot.

Given this consideration, the approach in Eq.(B.10) causes the non-exceeding torque components to be clamped down more than required, as they are reduced by the same scale-down factor. A short example to illustrate this:

$$\tau_s = \begin{pmatrix} 20 \\ 2 \\ 0 \end{pmatrix} \Rightarrow F = \begin{pmatrix} \mathbf{-14.26} \\ 50.10 \\ 64.36 \\ \mathbf{14.26} \\ -50.10 \\ -64.36 \end{pmatrix} \Rightarrow F_{si.max} = 30 \Rightarrow F^* = \begin{pmatrix} \mathbf{-6.65} \\ 23.35 \\ 30.00 \\ \mathbf{6.65} \\ -23.35 \\ -30.00 \end{pmatrix} \Rightarrow {\tau_s}^* = T \cdot F^* = \begin{pmatrix} 9.32 \\ 0.93 \\ 0.00 \end{pmatrix} \tag{B.11}$$

The torque applied around the y-axis is reduced to $\approx 46.26\%$, although not all stators do exceed their limit and can provide more thrust force.

The issue can be addressed by scaling the x- and y-component of the torque vector individually beforehand applying the approach in Eq.(B.10).

$$D = |T^+ \cdot I| \tag{B.12}$$

$$\tau_{s.max} = F_{si.max} / (\max_{1 \leq i \leq 3} d_{ij}) \tag{B.13}$$

$\tau_{s.max}$ contains the maximum permitted torque around x-, y- and z-axis, if the torque vector only has one non-zero component. Now, if one component in the torque vector τ_s is larger than its corresponding $\tau_{s.max}$ component, it is scaled down individually. After this, the obtained τ_s^{**} is tested as shown before in Eq.(B.10). A recalculation of the clamped down output using the modified approach from Eq.(B.12) yields:

$$\tau_s = \begin{pmatrix} 20 \\ 2 \\ 0 \end{pmatrix} \Rightarrow \tau_s^{**} = \begin{pmatrix} 9.72 \\ 2 \\ 0 \end{pmatrix} \Rightarrow F = \begin{pmatrix} \mathbf{-10.37} \\ 22.32 \\ 32.69 \\ \mathbf{10.37} \\ -22.32 \\ -32.69 \end{pmatrix} \Rightarrow F_{si.max} = 30 \Rightarrow F^* = \begin{pmatrix} \mathbf{-9.5168} \\ 20.4832 \\ 30.0000 \\ \mathbf{9.5168} \\ -20.4832 \\ -30.0000 \end{pmatrix}$$

$$\Rightarrow {\tau_s}^* = T \cdot F^* = \begin{pmatrix} 8.93 \\ 1.84 \\ 0 \end{pmatrix} \tag{B.14}$$

By using the second approach, the torque around the y-axis only got reduced to 91.78%.

B.2. Odometry Sensors

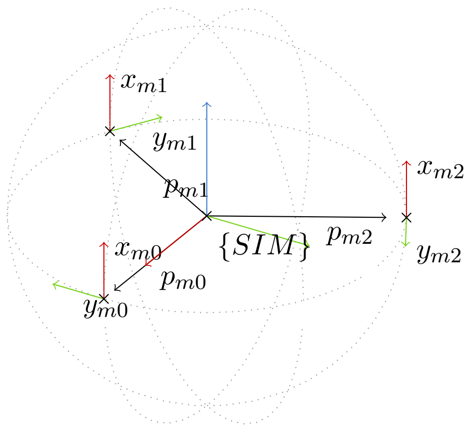

Figure 51: Sensor positions for $n_m = 3$

First, the three ($n_m = 3$) mouse sensor positions vectors p_{mi} ($i = 0, .., n_m - 1$) are defined in the spherical induction motor frame, where r_w is the spherical induction rotor wheel radius:

$$p_{mi} = r_w \begin{pmatrix} \cos(2\pi/n_m \cdot i) \\ -\sin(2\pi/n_m \cdot i) \\ 0 \end{pmatrix} \tag{B.15}$$

As can be seen from Fig 51 and Eq.(B.15), all n_m sensors are assumed in the equatorial plane of the rotor.

Next, the x_{mi} and y_{mi} unit vectors are defined, in which the surface velocity components are measured by the mouse sensors. While y_{mi} is different for each sensor, x_{mi} is the same as they share the same x-component-axis:

$$x_{mi} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad y_{mi} = \begin{pmatrix} -\sin(2\pi/n_m \cdot i) \\ -\cos(2\pi/n_m \cdot i) \\ 0 \end{pmatrix} \tag{B.16}$$

Given the angular velocity vector ω_s, the tangential surface velocity v_{mi} at the mouse sensor position p_{mi} can be calculated with the vector cross product:

$$v_{mi} = \omega_s \times p_{mi} \tag{B.17}$$

The components of v_{mi} measured by the sensor are:

$$v_{xmi} = \frac{v_{mi} \cdot x_{mi}}{|x_{mi}|} \qquad v_{ymi} = \frac{v_{mi} \cdot y_{mi}}{|y_{mi}|} \tag{B.18}$$

$$v_{xmi} = (\omega_s \times p_{mi}) \cdot x_{mi} \qquad v_{ymi} = (\omega_s \times p_{mi}) \cdot y_{mi} \tag{B.19}$$

The triple product shown in Eq.(B.19) can be rewritten to have ω_s included with the dot operator instead:

$$v_{xmi} = (p_{mi} \times x_{mi}) \cdot \omega_s \qquad v_{ymi} = (p_{mi} \times y_{mi}) \cdot \omega_s \tag{B.20}$$

Next, the dot-product for each individual component in Eq.(B.20) can be rewritten to a matrix notation:

$$\underbrace{\begin{pmatrix} v_{xm0} \\ v_{ym0} \\ \vdots \\ \vdots \end{pmatrix}}_{V_m} = \underbrace{\begin{pmatrix} (p_{m0} \times x_{m0})^T \\ (p_{m0} \times y_{m0})^T \\ \vdots \\ \vdots \end{pmatrix}}_{S \text{ ... sensor matrix}} \cdot \underbrace{\begin{pmatrix} \omega_{sx} \\ \omega_{sy} \\ \omega_{sz} \end{pmatrix}}_{\omega_s} \tag{B.21}$$

The matrix S on the right hand side of Eq.(B.21) is named sensor matrix and describes the sensor geometry, while the column vector V_m on the left hand side consists of the $2 \cdot n_m$ mouse sensor surface velocity components. Eq.(B.21) shows the relation between the sensor surface velocities (V_m) and ω_s. However, it yields the sensor surface velocities for a known ω_s, whereas the opposite is actually needed.

As in the previous section 3.2, again the left pseudo inverse is applied, this time to S, to solve for the angular velocity ω_s:

$$S^+ \cdot V_m = \omega_s. \tag{B.22}$$

The inverse sensor matrix S^+ is time-independent, as it only contains the sensor position information and their axis alignments.

For a $n_m = 3$ configuration, as it is used on the spherical induction motor, the following result can be retrieved for S^+:

$$S^+_{n_m=3} = \frac{1}{r_w} \cdot \begin{pmatrix} 0 & 0 & -\frac{\sqrt{3}}{3} & 0 & \frac{\sqrt{3}}{3} & 0 \\ -\frac{2}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & -\frac{1}{3} & 0 & -\frac{1}{3} & 0 & -\frac{1}{3} \end{pmatrix} \tag{B.23}$$

The surface velocity y-component axis y_{mi} of each sensor points in a different direction, but they share the same rotation axis (the z-axis of the spherical induction motor coordinate frame). Therefore, as can be seen in Eq.(B.23), the solution $S^+_{n_m=3}$ takes an evenly weighted average of the three components y_{m0}, y_{m1} and y_{m2}.

C. 2D Planar Model

C.1. External Disturbance

Later on it might be useful to study the behavior of ballbot when exhibited to an external disturbance. The external disturbance is represented by an external force F_e, applied to the upper body of the robot at height h_e. This force F_e could, for example, represent a person pushing the robot.

So far no external forces have been considered in the dynamic model. The external force is added in the same fashion as it has been done for the drive torque in section 4.6.The external force F_e and the corresponding position vector v_e, both in Cartesian coordinates, are:

$$F_e = \begin{pmatrix} F \\ 0 \\ 0 \end{pmatrix} \qquad v_e = v_w + h_e \cdot \begin{pmatrix} \tan(\phi) \\ 0 \\ 1 \end{pmatrix} \tag{C.1}$$

In the next step F_e is transformed from Cartesian coordinates to the generalized coordinates q:

$$F_{eq} = \left(\frac{\partial v_e}{\partial q}\right)^T \cdot F_e = \begin{pmatrix} r_w & h_e \sec(\phi)^2 + r_w \\ 0 & 0 \\ 0 & 0 \end{pmatrix}^T \cdot F_e = \begin{pmatrix} r_w F \\ \left(h_e \sec(\phi)^2 + r_w\right) F \end{pmatrix} \tag{C.2}$$

F_{eq} is added to the external forces vector Q. All other terms of the motion equation remain unaffected. The new solutions for $\ddot{\theta}$ and $\ddot{\phi}$, both denoted with index e, are:

$$\ddot{\theta}_e = \ddot{\theta} + \frac{h_e\left(\beta\cos(\phi) + \alpha\right) - r_w \cos(\phi)^2 \left(\beta\cos(\phi) + \gamma\right)}{\cos(\phi)^2 \left(\beta^2 \cos(\phi)^2 - \gamma\,\alpha\right)} \cdot F \tag{C.3}$$

$$\ddot{\phi}_e = \ddot{\phi} + \frac{\left(r_w \beta \cos(\phi)^3 - \alpha\, h_e\right)}{\cos(\phi)^2 \left(\beta^2 \cos(\phi)^2 - \gamma\,\alpha\right)} \cdot F \tag{C.4}$$

To include the new terms introduced by the external force in the state space representation, both have to be linearized (as done before in section 4.8):

$$\ddot{\theta}_e = \ddot{\theta} + \frac{((\beta+\alpha)\ h_e - r_w\,\beta - \gamma\, r_w)\ F}{\beta^2 - \gamma\,\alpha} \tag{C.5}$$

$$\ddot{\phi}_e = \ddot{\phi} - \frac{(\alpha\, h_e - r_w\,\beta)\ F}{\beta^2 - \gamma\,\alpha} \tag{C.6}$$

The disturbance is commonly included in the state space system as matrix E in addition to A and B:

$$\dot{x}(t) = A\,x(t) + B\,u(t) + E\,d(t) \tag{C.7}$$

$$y(t) = C\,x(t) + D\,u(t) \tag{C.8}$$

Taking the linearized results from Eq.(C.5) yields the following matrix E and vector d:

$$E = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{((\beta+\alpha)\,h_e - r_w\,\beta - \gamma\,r_w)\,F}{\beta^2-\gamma\,\alpha} & 0 \\ 0 & 0 & 0 & -\frac{(\alpha\,h_e - r_w\,\beta)\,F}{\beta^2-\gamma\,\alpha} \end{pmatrix} \qquad d(t) = \begin{pmatrix} 0 \\ 0 \\ F(t) \\ F(t) \end{pmatrix} \tag{C.9}$$

D. Maxima Scripts

The scripts have been created in the wxmaxima viewer. The output formatting defines are specific to wxmaxima, but not required to run the scripts. The defines just create an easier readable formatting for the variable symbols.

D.1. Lagrange Equations Rev2

output formatting defines

```
(%i1) :lisp (defprop $lb "<i>l<n>b</n></i>" wxxmlword);

(%i1) :lisp (defprop $mb "<i>m<n>b</n></i>" wxxmlword);

(%i1) :lisp (defprop $mw "<i>m<n>w</n></i>" wxxmlword);

(%i1) :lisp (defprop $iw "<i>I<n>w</n></i>" wxxmlword);

(%i1) :lisp (defprop $ib "<i>I<n>b</n></i>" wxxmlword);

(%i1) :lisp (defprop $th "<g>theta</g>" wxxmlword);

(%i1) :lisp (defprop $ph "<g>phi</g>" wxxmlword);

(%i1) :lisp (defprop $rw "<i>r<n>w</n></i>" wxxmlword);

(%i1) :lisp (defprop $he "<i>h<n>e</n></i>" wxxmlword);

(%i1) :lisp (defprop $xw "<i>x<n>w</n></i>" wxxmlword);
```

```
(%i1) :lisp (defprop $xb "<i>x<n>b</n></i>" wxxmlword);
```

Euler-Lagrange equations

Variable definitions and assumptions

```
(%i1) declare([ph,th,t], real)$
```

```
(%i2) declare([lb,mb,mw,rw, ib, iw,g], [real, constant])$
```

```
(%i3) assume([lb,mb,mw,rw, ib, iw,g] > 0)$
```

```
(%i4) depends([ph,th],t);
```

(%o4) $[\phi(t), \theta(t)]$

Generalized Coordinates:

```
(%i5) q : [th, ph];
```

(%o5) $[\theta, \phi]$

```
(%i6) facts(ph);
```

(%o6) $[\mathrm{kind}(\phi, real)]$

```
(%i7) dph: diff(ph,t);
```

(%o7) $\frac{d}{d\,t}\,\phi$

```
(%i8) dth: diff(th,t);
```

(%o8) $\frac{d}{d\,t}\,\theta$

```
(%i9) ddth: diff(dth,t);
```

(%o9) $\frac{d^2}{d\,t^2}\,\theta$

```
(%i10) ddph: diff(dph,t);
```

(%o10) $\frac{d^2}{d\,t^2}\,\phi$

```
(%i11) dq :  [dth, dph];
```

(%o11) $[\frac{d}{d\,t}\,\theta,\frac{d}{d\,t}\,\phi]$

Wheel

mass center position vector:

```
(%i12) v[w] : matrix([(ph+th)*(rw)], [0],[0]);
```

(%o12) $\begin{pmatrix} r_w\,\left(\theta+\phi\right) \\ 0 \\ 0 \end{pmatrix}$

```
(%i13) dv[w] : diff(v[w],t);
```

(%o13) $\begin{pmatrix} r_w\,\left(\frac{d}{d\,t}\,\theta+\frac{d}{d\,t}\,\phi\right) \\ 0 \\ 0 \end{pmatrix}$

kinetic energy:

```
(%i14) T_old[w] := 1/2 * mw * (rw * dph)^2 + 1/2 * iw * (dph)^2;
```

(%o14) $T\_old_w := \frac{1}{2}\,m_w\,\left(r_w\,dph\right)^2+\frac{1}{2}\,I_w\,dph^2$

```
(%i15) T[w] : facsum(1/2*mw * trigsimp(dv[w]. dv[w]) +  1/2*iw * (dph+dth)^2);
```

(%o15) $\frac{\left(m_w\,{r_w}^2+I_w\right)\,\left(\frac{d}{d\,t}\,\theta+\frac{d}{d\,t}\,\phi\right)^2}{2}$

potential energy:

```
(%i16) V[w]  : 0;
```

(%o16) 0

Body

mass center position vector:

```
(%i17) v[b] : (v[w] + lb*matrix([sin(ph)], [0],[cos(ph)]));
```

(%o17) $$\begin{pmatrix} r_w\ (\theta+\phi)+l_b\sin(\phi) \\ 0 \\ l_b\cos(\phi) \end{pmatrix}$$

```
(%i18) dv[b] : diff(v[b],t);
```

(%o18) $$\begin{pmatrix} r_w\ \left(\frac{d}{d\,t}\,\theta+\frac{d}{d\,t}\,\phi\right)+l_b\cos(\phi)\ \left(\frac{d}{d\,t}\,\phi\right) \\ 0 \\ -l_b\sin(\phi)\ \left(\frac{d}{d\,t}\,\phi\right) \end{pmatrix}$$

potential energy:

```
(%i19) V[b]  : mb * g * v[b][3][1];
```

(%o19) $g\,l_b\,m_b\cos(\phi)$

kinetic energy:

```
(%i20) T[b] : facsum( 1/2*mb * trigsimp(dv[b]. dv[b]) +  1/2*ib * (dph)^2
       ,ddph,ddth,dth,dph);
```

(%o20) $$\frac{m_b\,{r_w}^2\left(\frac{d}{d\,t}\,\theta\right)^2+2\,m_b\,r_w\ (l_b\cos(\phi)+r_w)\ \left(\frac{d}{d\,t}\,\phi\right)\ \left(\frac{d}{d\,t}\,\theta\right) + \left(2\,l_b\,m_b\,r_w\cos(\phi)+m_b\,{r_w}^2+{l_b}^2\,m_b+I_b\right)\ \left(\frac{d}{d\,t}\,\phi\right)^2}{2}$$

Total Energy

Potential energy:

```
(%i21) V : V[b] + V[w];
```

(%o21) $g\,l_b\,m_b\cos(\phi)$

Kinetic Energy:

```
(%i22) Tt: facsum(T[b] + T[w],ddph,ddth,dth,dph);
```

(%o22) $$((m_w r_w^2+m_b r_w^2+I_w)\left(\frac{d}{dt}\theta\right)^2+2\left(l_b m_b r_w\cos(\phi)+m_w r_w^2+m_b r_w^2+I_w\right)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)+\left(2 l_b m_b r_w\cos(\phi)+m_w r_w^2+m_b r_w^2+l_b^2 m_b+I_w+I_b\right)\left(\frac{d}{dt}\phi\right)^2)/2$$

substitute constant expressions with alpha, beta, gamma:

```
(%i23) Tt: ratsubst(%beta, lb*mb*rw,  Tt);
```

(%o23) $$(((m_w+m_b) r_w^2+I_w)\left(\frac{d}{dt}\theta\right)^2+\left(2\beta\cos(\phi)+(2 m_w+2 m_b) r_w^2+2 I_w\right)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)+\left(2\beta\cos(\phi)+(m_w+m_b) r_w^2+l_b^2 m_b+I_w+I_b\right)\left(\frac{d}{dt}\phi\right)^2)/2$$

```
(%i24) Tt: ratsubst(%gamma, ib+lb^2 * mb,  Tt);
```

(%o24) $$(((m_w+m_b) r_w^2+I_w)\left(\frac{d}{dt}\theta\right)^2+\left(2\beta\cos(\phi)+(2 m_w+2 m_b) r_w^2+2 I_w\right)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)+\left(2\beta\cos(\phi)+(m_w+m_b) r_w^2+I_w+\gamma\right)\left(\frac{d}{dt}\phi\right)^2)/2$$

```
(%i25) Tt: ratsubst(%alpha, mb*rw^2 + mw*rw^2 +iw,  Tt);
```

(%o25) $$\frac{\alpha\left(\frac{d}{dt}\theta\right)^2+(2\beta\cos(\phi)+2\alpha)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)+(2\beta\cos(\phi)+\alpha+\gamma)\left(\frac{d}{dt}\phi\right)^2}{2}$$

```
(%i26) V: ratsubst(%epsilon, g*lb*mb,  V);
```

(%o26) $\epsilon\cos(\phi)$

Non-potential/external forces

angular velocity vector for the wheel (wheel and body located the XZ-plane):

```
(%i27) %omega[w] : matrix([0],[-dph+dth],[0]);
```

(%o27) $$\begin{pmatrix} 0 \\ \frac{d}{dt}\theta - \frac{d}{dt}\phi \\ 0 \end{pmatrix}$$

```
(%i28) %omega[b] : matrix([0],[-dph],[0]);
```

(%o28) $$\begin{pmatrix} 0 \\ -\frac{d}{dt}\phi \\ 0 \end{pmatrix}$$

applied torque:

```
(%i29) To[w]: matrix([0],[Tsim],[0]);
```

(%o29) $$\begin{pmatrix} 0 \\ Tsim \\ 0 \end{pmatrix}$$

```
(%i30) To[b]: -To[w];
```

(%o30) $$\begin{pmatrix} 0 \\ -Tsim \\ 0 \end{pmatrix}$$

required in respect to the generalized coordinates, transformation matrix:

```
(%i31) J[R1] : jacobian(flatten(args(%omega[w])), dq);
```

(%o31) $$\begin{pmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{pmatrix}$$

```
(%i32) J[R2] : jacobian(flatten(args(%omega[b])), dq);
```

(%o32) $$\begin{pmatrix} 0 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$$

```
(%i33) F[NP] : transpose(J[R1]) . To[w] + transpose(J[R2]) . To[b];
```

(%o33) $$\begin{pmatrix} Tsim \\ 0 \end{pmatrix}$$

Euler-Lagrange equation

```
(%i34) dV: transpose(jacobian([V],q));
```

$$(\%o34)\quad \begin{pmatrix} 0 \\ -\epsilon \sin(\phi) \end{pmatrix}$$

```
(%i35) dT: transpose(jacobian([Tt],q));
```

$$(\%o35)\quad \begin{pmatrix} 0 \\ \frac{-2\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)-2\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2}{2} \end{pmatrix}$$

```
(%i36) ddT: facsum(fullratsimp(diff(transpose(jacobian([Tt],dq)),t))

       ,ddph,ddth,dth,dph);
```

$$(\%o36)\quad \begin{pmatrix} \alpha\left(\frac{d^2}{dt^2}\theta\right)+(\beta\cos(\phi)+\alpha)\left(\frac{d^2}{dt^2}\phi\right)-\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2 \\ \left[\begin{matrix}(\beta\cos(\phi)+\alpha)\left(\frac{d^2}{dt^2}\theta\right)-\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)\left(\frac{d}{dt}\theta\right)+(2\beta\cos(\phi)+\alpha+\gamma)\left(\frac{d^2}{dt^2}\phi\right) \\ -2\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2\end{matrix}\right] \end{pmatrix}$$

```
(%i37) L1: facsum(ddT-dT,  ddph,ddth,dth,dph);
```

$$(\%o37)\quad \begin{pmatrix} \alpha\left(\frac{d^2}{dt^2}\theta\right)+(\beta\cos(\phi)+\alpha)\left(\frac{d^2}{dt^2}\phi\right)-\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2 \\ (\beta\cos(\phi)+\alpha)\left(\frac{d^2}{dt^2}\theta\right)+(2\beta\cos(\phi)+\alpha+\gamma)\left(\frac{d^2}{dt^2}\phi\right)-\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2 \end{pmatrix}$$

```
(%i38) M: transpose(matrix(flatten(args(coeff(L1, ddth))),

       flatten(args(coeff(L1,ddph)))));
```

$$(\%o38)\quad \begin{pmatrix} \alpha & \beta\cos(\phi)+\alpha \\ \beta\cos(\phi)+\alpha & 2\beta\cos(\phi)+\alpha+\gamma \end{pmatrix}$$

```
(%i39) C : L1 - M . matrix([ddth], [ddph]);
```

$$(\%o39)\quad \begin{pmatrix} -\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2 \\ -\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2 \end{pmatrix}$$

```
(%i40) eq1: (L1 + dV - F[NP])[1][1] = 0;
```

$$(\%o40)\quad -Tsim+\alpha\left(\frac{d^2}{dt^2}\theta\right)+(\beta\cos(\phi)+\alpha)\left(\frac{d^2}{dt^2}\phi\right)-\beta\sin(\phi)\left(\frac{d}{dt}\phi\right)^2=0$$

```
(%i41) eq2: (L1 + dV - F[NP])[2][1] = 0;
```

(%o41) $$(\beta\cos(\phi)+\alpha)\left(\frac{d^2}{d\,t^2}\,\theta\right)+(2\,\beta\cos(\phi)+\alpha+\gamma)\left(\frac{d^2}{d\,t^2}\,\phi\right)-\beta\sin(\phi)\left(\frac{d}{d\,t}\,\phi\right)^2-\epsilon\sin(\phi)=0$$

```
(%i42) sol: solve([eq1,eq2],[ddth, ddph]);
```

(%o42) $$[[\frac{d^2}{d\,t^2}\,\theta=-\frac{(2\,\beta\cos(\phi)+\alpha+\gamma)\;Tsim+\left(\beta^2\cos(\phi)+\gamma\,\beta\right)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2+(-\beta\,\epsilon\cos(\phi)-\alpha\,\epsilon)\,\sin(\phi)}{\beta^2\cos(\phi)^2-\gamma\,\alpha},$$
$$\frac{d^2}{d\,t^2}\,\phi=\frac{(\beta\cos(\phi)+\alpha)\;Tsim+\beta^2\cos(\phi)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2-\alpha\,\epsilon\sin(\phi)}{\beta^2\cos(\phi)^2-\gamma\,\alpha}]]$$

```
(%i43) sol2: facsum(sol, dth,dph, th, ph, Tsim);
```

(%o43) $$[[\frac{d^2}{d\,t^2}\,\theta=\frac{-(2\,\beta\cos(\phi)+\alpha+\gamma)\;Tsim-\beta\,(\beta\cos(\phi)+\gamma)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2+\epsilon\,(\beta\cos(\phi)+\alpha)\,\sin(\phi)}{\beta^2\cos(\phi)^2-\gamma\,\alpha},$$
$$\frac{d^2}{d\,t^2}\,\phi=\frac{(\beta\cos(\phi)+\alpha)\;Tsim+\beta^2\cos(\phi)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2-\alpha\,\epsilon\sin(\phi)}{\beta^2\cos(\phi)^2-\gamma\,\alpha}]]$$

Linearisation

```
(%i44) lin1: ratsubst(1, cos(ph), sol2[1]);
```

(%o44) $$[\frac{d^2}{d\,t^2}\,\theta=-\frac{(2\,\beta+\alpha+\gamma)\;Tsim+\left(\beta^2+\gamma\,\beta\right)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2+(-\beta-\alpha)\,\epsilon\sin(\phi)}{\beta^2-\gamma\,\alpha},$$
$$\frac{d^2}{d\,t^2}\,\phi=\frac{(\beta+\alpha)\;Tsim+\beta^2\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2-\alpha\,\epsilon\sin(\phi)}{\beta^2-\gamma\,\alpha}]$$

```
(%i45) lin2: ratsubst(ph, sin(ph), lin1);
```

(%o45) $$[\frac{d^2}{d\,t^2}\,\theta=-\frac{(2\,\beta+\alpha+\gamma)\;Tsim+\left(\beta^2\,\phi+\gamma\,\beta\,\phi\right)\,\left(\frac{d}{d\,t}\,\phi\right)^2+\epsilon\,(-\beta\,\phi-\alpha\,\phi)}{\beta^2-\gamma\,\alpha},$$
$$\frac{d^2}{d\,t^2}\,\phi=\frac{(\beta+\alpha)\;Tsim+\beta^2\,\phi\,\left(\frac{d}{d\,t}\,\phi\right)^2-\alpha\,\epsilon\,\phi}{\beta^2-\gamma\,\alpha}]$$

```
(%i46) lin3: ratsubst(0, dph, lin2);
```

(%o46) $$[\frac{d^2}{d\,t^2}\,\theta=-\frac{\left(2\,\beta+\alpha+\gamma\right)\,Tsim+\left(-\beta-\alpha\right)\,\epsilon\,\phi}{\beta^2-\gamma\,\alpha},\frac{d^2}{d\,t^2}\,\phi=\frac{\left(\beta+\alpha\right)\,Tsim-\alpha\,\epsilon\,\phi}{\beta^2-\gamma\,\alpha}]$$

External Force

```
(%i47) F[e]: matrix([F], [0],[0]);
```

(%o47) $$\begin{pmatrix}F\\0\\0\end{pmatrix}$$

```
(%i48) v[e]: v[w] + he* matrix([tan(ph)], [0],[0]);
```

(%o48) $$\begin{pmatrix}r_w\,\left(\theta+\phi\right)+h_e\,\tan\left(\phi\right)\\0\\0\end{pmatrix}$$

```
(%i49) J[Fe] : jacobian(flatten(args(v[e])), q);
```

(%o49) $$\begin{pmatrix}r_w & h_e\,\sec\left(\phi\right)^2+r_w\\0&0\\0&0\end{pmatrix}$$

```
(%i50) F[NP2]: F[NP] + transpose(J[Fe]).F[e];
```

(%o50) $$\begin{pmatrix}r_w\,F+Tsim\\\left(h_e\,\sec\left(\phi\right)^2+r_w\right)\,F\end{pmatrix}$$

```
(%i51) eqF1: (L1 + dV - F[NP2])[1][1] = 0;
```

(%o51) $$-r_w\,F-Tsim+\alpha\,\left(\frac{d^2}{d\,t^2}\,\theta\right)+\left(\beta\,\cos\left(\phi\right)+\alpha\right)\,\left(\frac{d^2}{d\,t^2}\,\phi\right)-\beta\,\sin\left(\phi\right)\,\left(\frac{d}{d\,t}\,\phi\right)^2=0$$

```
(%i52) eqF2: (L1 + dV - F[NP2])[2][1] = 0;
```

(%o52) $$-\left(h_e\,\sec\left(\phi\right)^2+r_w\right)\,F+\left(\beta\,\cos\left(\phi\right)+\alpha\right)\,\left(\frac{d^2}{d\,t^2}\,\theta\right)+\left(2\,\beta\,\cos\left(\phi\right)+\alpha+\gamma\right)\,\left(\frac{d^2}{d\,t^2}\,\phi\right)-\beta\,\sin\left(\phi\right)\,\left(\frac{d}{d\,t}\,\phi\right)^2-\epsilon\,\sin\left(\phi\right)=0$$

```
(%i53) solF: solve([eqF1,eqF2],[ddth, ddph]);
```

(%o53) $$[[\frac{d^2}{d\,t^2}\,\theta=-((\beta\cos(\phi)\left(r_w-h_e\sec(\phi)^2\right)-\alpha\,h_e\sec(\phi)^2+\gamma\,r_w)\;F$$

$$+(2\,\beta\cos(\phi)+\alpha+\gamma)\;Tsim+\left(\beta^2\cos(\phi)+\gamma\,\beta\right)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2$$

$$+(-\beta\,\epsilon\cos(\phi)-\alpha\,\epsilon)\,\sin(\phi))/(\beta^2\cos(\phi)^2-\gamma\,\alpha),\frac{d^2}{d\,t^2}$$

$$\phi=\frac{\left(r_w\,\beta\cos(\phi)-\alpha\,h_e\sec(\phi)^2\right)\,F+(\beta\cos(\phi)+\alpha)\;Tsim+\beta^2\cos(\phi)\,\sin(\phi)\,\left(\frac{d}{d\,t}\,\phi\right)^2-\alpha\,\epsilon\sin(\phi)}{\beta^2\cos(\phi)^2-\gamma\,\alpha}]]$$

```
(%i54) addterm1: facsum(trigsimp(expand(rhs(solF[1][1]) - rhs(sol2[1][1]))),F,rw,he);
```

(%o54) $$\frac{h_e\,(\beta\cos(\phi)+\alpha)\;F-r_w\cos(\phi)^2\,(\beta\cos(\phi)+\gamma)\;F}{\cos(\phi)^2\,\left(\beta^2\cos(\phi)^2-\gamma\,\alpha\right)}$$

```
(%i55) addterm2: facsum(trigsimp(expand(rhs(solF[1][2]) - rhs(sol2[1][2]))),F);
```

(%o55) $$\frac{\left(r_w\,\beta\cos(\phi)^3-\alpha\,h_e\right)\;F}{\cos(\phi)^2\,\left(\beta^2\cos(\phi)^2-\gamma\,\alpha\right)}$$

```
(%i56) linF1: ratsubst(1, cos(ph), addterm1);
```

(%o56) $$\frac{((\beta+\alpha)\;h_e-r_w\,\beta-\gamma\,r_w)\;F}{\beta^2-\gamma\,\alpha}$$

```
(%i57) linF2: ratsubst(1, cos(ph), addterm2);
```

(%o57) $$-\frac{(\alpha\,h_e-r_w\,\beta)\;F}{\beta^2-\gamma\,\alpha}$$

References

[1] Microdynamic Systems Laboratory an der Carnegie Mellon University. Dynamically-stable mobile robots in human environments. http://www.msl.ri.cmu.edu/projects/ballbot/. Accessed August 24, 2015.

[2] Kameshwar Poolla Andrew Packard and Roberto Horowitz. Dynamic systems and feedback class notes for me 132 (chapters 19-23. linear systems). http://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/pph.html. Accessed August 24, 2015.

[3] Masaaki Kumagai Ankit Bhatia and Ralph Hollis. Six-stator spherical induction motor for balancing mobile robots. *IEEE Int'l. Conf. on Robotics and Automation*, 2015.

[4] Chris Atkeson. Team wpi-cmu: Darpa robotics challenge. http://www.cs.cmu.edu/~cga/drc/. Accessed August 28, 2015.

[5] Johannes Auer and Olaf Sassnick. Entwicklung und Bau eines Ballbots. Pre Bachelor Work, Salzburg University of Applied Sciences, 2013.

[6] Adi Ben-Israel and Thomas N.E. Greville. *Generalized inverses: Theory and applications*. Springer-Verlag, New York, 2nd edition, 2003.

[7] Panasonic Corporation. http://news.panasonic.com/global/topics/2015/44009.html. http://news.panasonic.com/global/topics/2015/44009.html. Accessed August 24, 2015.

[8] DARPA. Darpa robotics challenge finals 2015. http://www.theroboticschallenge.org. Accessed August 24, 2015.

[9] ODE Developers. Bitbucket ode - commits. https://bitbucket.org/odedevs/ode/commits/all. Accessed August 24, 2015.

[10] Stephen J. Dodds. *Feedback Control*. Springer-Verlag, London, 2015.

[11] J. Ginsberg. *Engineering Dynamics*. Cambridge University Press, Cambridge, 2007.

[12] Erico Guizzo. A robot that balances on a ball. `http://spectrum.ieee.org/automaton/robotics/robotics-software/042910-a-robot-that-balances-on-a-ball`. Accessed August 24, 2015.

[13] PixArt Imaging Inc. Adns 9800 - data sheet. `http://www.pixart.com.tw/upload/ADNS-9800%20DS_S_V1.0_20130514144352.pdf`. Accessed August 24, 2015.

[14] Fraunhofer IPA. Care-o-bot 4. `http://www.care-o-bot.de/en/care-o-bot-4.html`. Accessed August 24, 2015.

[15] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 2002.

[16] K Kaneko, I Yamada, and K Itao. A spherical dc servo motor with three degrees of freedom. *Journal of dynamic systems, measurement, and control*, 111(3):398–402, 1989.

[17] Tom Lauwers, George Kantor and Ralph Hollis. One is enough! *Proc. Int'l. Symp. for Robotics Research*, 2005.

[18] Masaaki Kumagai. Ballipsm - ballip snowman, balancing robot with double ball. `https://www.youtube.com/watch?v=ZjlZN9qhTXY`. Accessed August 28, 2015.

[19] Masaaki Kumagai and Ralph Hollis. Development and control of a three dof planar induction motor. *IEEE Int'l. Conf. on Robotics and Automation*, 2012.

[20] Masaaki Kumagai and Ralph L Hollis. Development of a three-dimensional ball rotation sensing system using optical mouse sensors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5038–5043. IEEE, 2011.

[21] Masaaki Kumagai and Ralph L Hollis. Development and control of a three dof spher-

ical induction motor. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1528–1533. IEEE, 2013.

[22] TB Lauwers, George A Kantor, and Ralph L Hollis. A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2884–2889. IEEE, 2006.

[23] Kok-Meng Lee, Hungsun Son, Jeffry Joni, et al. Concept development and design of a spherical wheel motor (SWM). In *IEEE International Conference on Robotics and Automation*, volume 4, page 3652. IEEE; 1999, 2005.

[24] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research, 15(Nov):3735–3739*, 2014.

[25] International Federation of Robotics. Industrial robot statistics. `http://www.ifr.org/industrial-robots/statistics/`. Accessed August 24, 2015.

[26] International Federation of Robotics. Service robot. `http://www.ifr.org/service-robots/`. Accessed August 24, 2015.

[27] International Federation of Robotics. Service robot statistics. `http://www.ifr.org/service-robots/statistics/`. Accessed August 24, 2015.

[28] Karl Johan Åstrom and Richard M. Murray. *Feedback Systems: An introduction for Scientists and Engineers*. Princeton University Press, New Jersey, 2008.

[29] Robocave. Darstellung eines mecanum wheels. `http://robocave.pk/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/1/0/100mm_mecanum_h.jpg`. Accessed August 24, 2015.

[30] RobotorNETZ. Omniwheels. `http://www.rn-wissen.de/index.php/OmniWheels`. Accessed August 24, 2015.

[31] Alison Sander and Meldon Wolfgang. The rise of robotics. https://www.bcgperspectives.com/content/articles/business_unit_strategy_innovation_rise_of_robotics/. Accessed August 24, 2015.

[32] Savioke. Savioke. http://www.savioke.com/. Accessed August 24, 2015.

[33] Michael Shomin and Ralph Hollis. Differentially flat trajectory generation for a dynamically stable mobile robot. *IEEE Int'l. Conf. on Robotics and Automation*, 2013.

[34] Michael Shomin and Ralph Hollis. Fast, dynamic trajectory planning for a dynamically stable mobile robot. *IEEE Int'l. Conf. on Robotics and Automation*, 2014.

[35] Shegeki Toyama, Shigeru Sugitani, Zhang Guoqiang, Yasutaro Miyatani, and Kazuto Nakamura. Multi degree of freedom spherical ultrasonic motor. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 2935–2940. IEEE, 1995.

[36] Tuxfamily. Eigen c++ template library. http://eigen.tuxfamily.org/index.php?title=Main_Page. Accessed August 24, 2015.

[37] Nagarajan Umashankar, George Kantor, and Ralph Hollis. The ballbot: An omnidirectional balancing mobile robot. *The International Journal of Robotics Research*, 33(6):917–930, May 2013.

[38] George Kantor Umashankar Nagarajan and Ralph Hollis. Integrated planning and control for graceful navigation of shape-accelerated underactuated balancing mobile robots. *IEEE Int'l. Conf. on Robotics and Automation*, 2012.

[39] Frankhauser P. und Gwerder C. Modeling and control of a ballbot. *ETH Zürich*, 2010.

[40] Wikipedia. Omniwheel. http://en.wikipedia.org/wiki/Omni_wheel. Accessed August 24, 2015.

[41] hosted by the VDMA Worldrobotics, IFR Statistical Department. Executive summary: World robotics 2014 1. industrial robots, 2. service robots. `http://www.worldrobotics.org/uploads/media/Executive_Summary_WR_2014_02.pdf`. Accessed August 28, 2015.

[42] ETH Zürich. Rezero projekt. `http://rezero.ethz.ch`. Accessed August 24, 2015.