MCECS Bot Developer Guide

Summer 2014, revision 1.1

Table of Contents

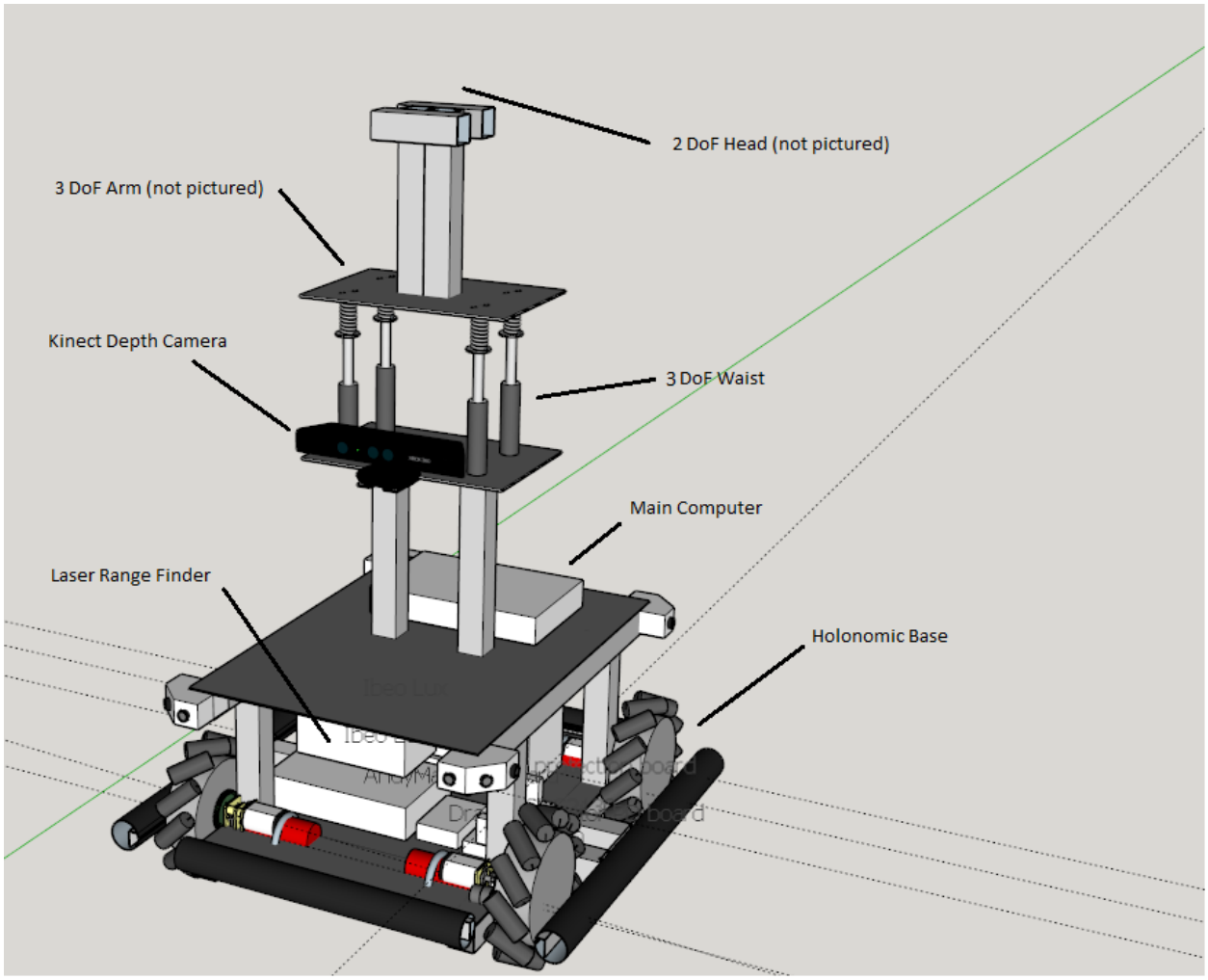
Introduction

This document is a guide to developing software applications for the Portland State MCECS Bot, also known as Jeeves. Additionally, this document provides basic operating and testing procedures for the robot, and pointers to additional documentation.

The intended audience is PSU students working in Dr. Perkowski's Intelligent Robotics Lab who need to work with Jeeves to complete coursework or conduct robotics research.

Jeeves Overview

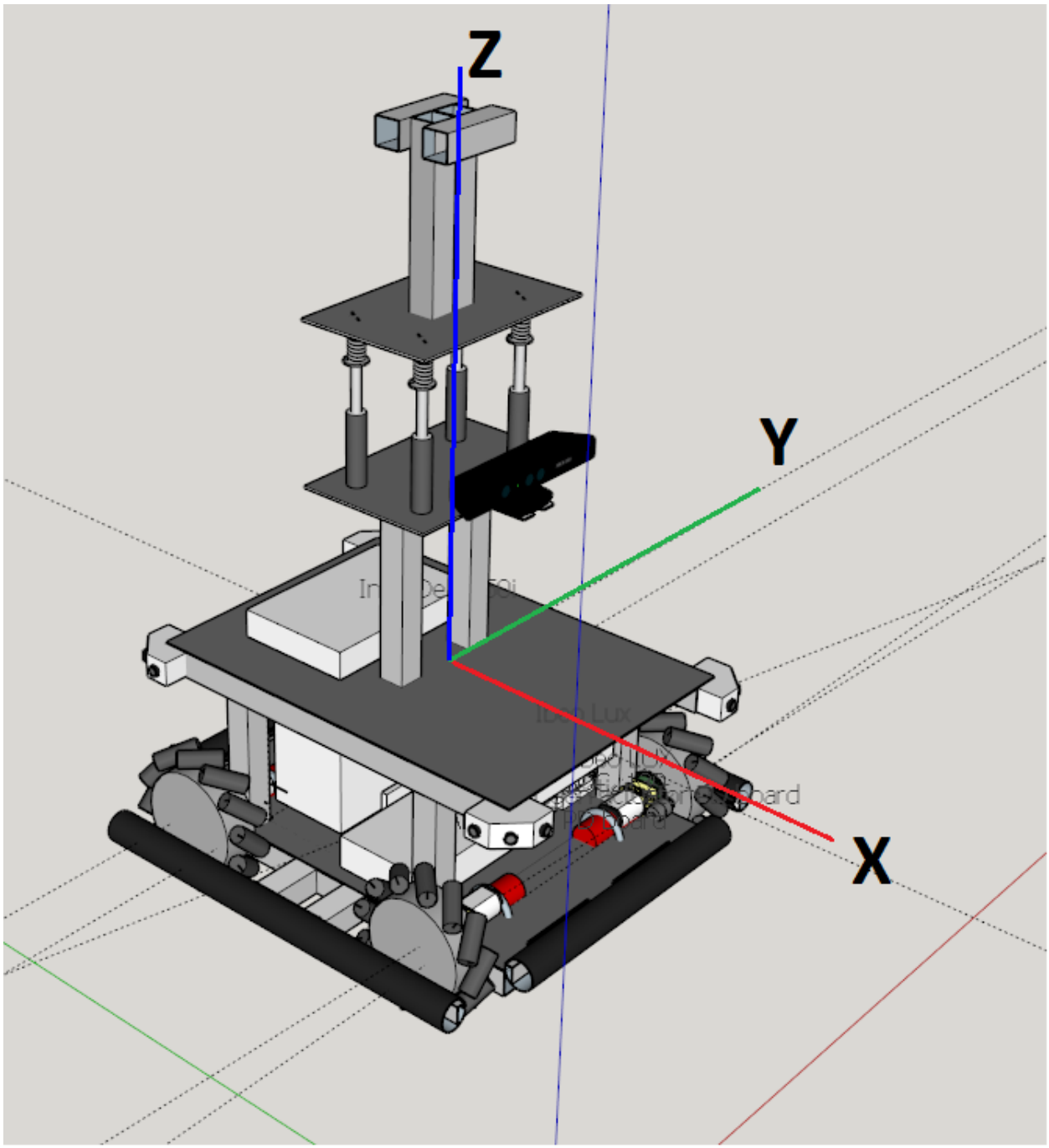
Jeeves is a humanoid mobile robot base with an array of actuators and sensors:



Actuators

Holonomic Base

The base features Mecanum wheels; the base can be actuated simultaneously in the x, y (translation), and z axes (rotation):



Motion Model

Figure 1 depicts the forces acting on a Mecanum wheel vehicle [1] [2]:

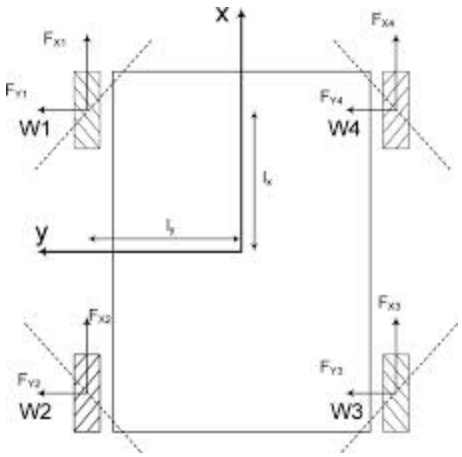


Fig. 1

The transformation from wheel velocities to base velocity is calculated by the following:

$$[\frac{R}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ \frac{-1}{L} & \frac{-1}{L} & \frac{1}{L} & \frac{1}{L} \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

where
R = 0.1016 m (wheel radius)
L = l_x + l_y = 0.2032 m + 0.2667 m (half wheelbase x and y)
w_1 = angular velocity of wheel 1
w_2 = angular velocity of wheel 2
w_3 = angular velocity of wheel 3
w_4 = angular velocity of wheel 4

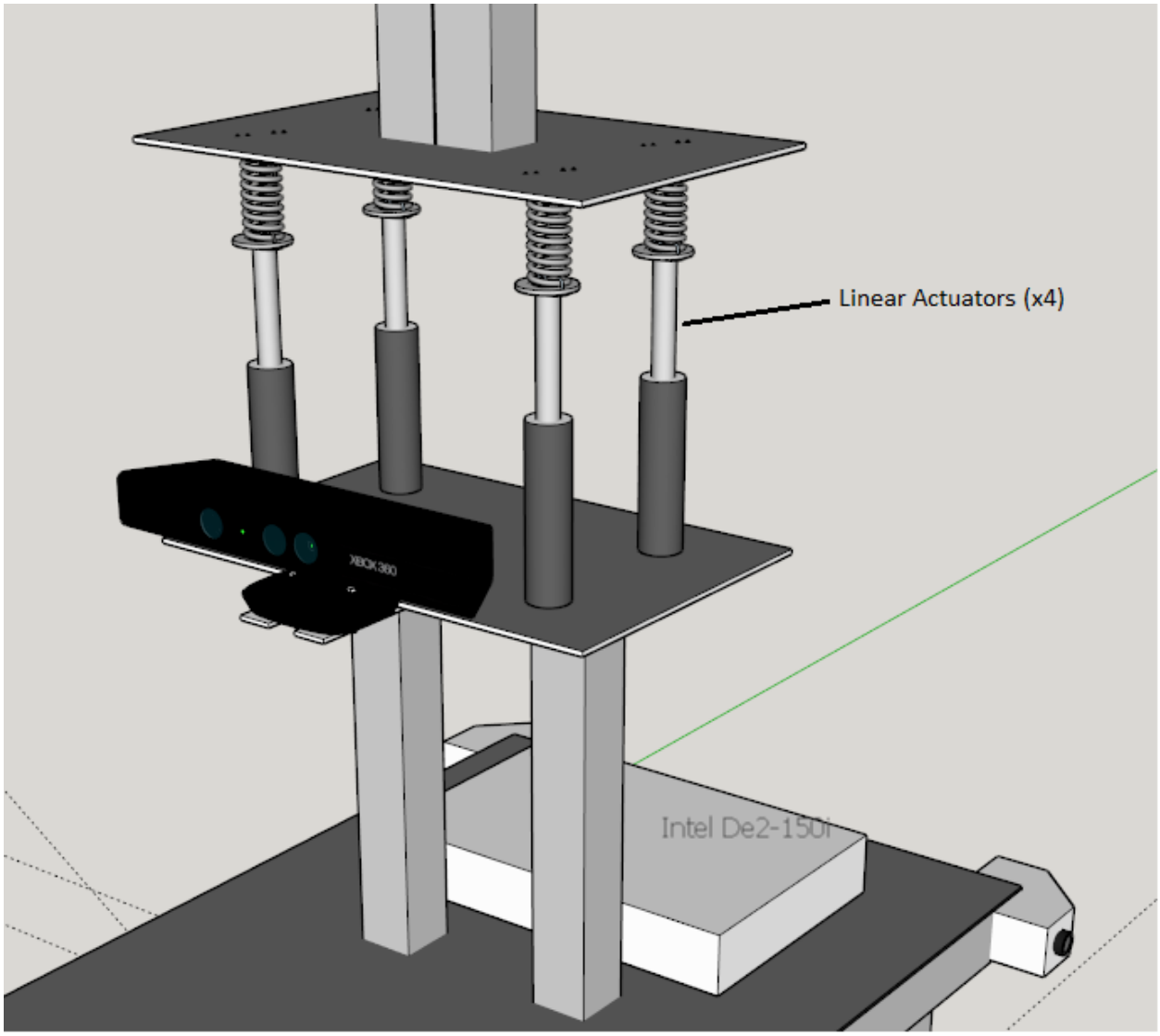
The inverse transformation is given by

$$\frac{1}{R}\begin{pmatrix} 1 & -1 & -L \\ 1 & 1 & -L \\ 1 & -1 & L \\ 1 & 1 & L \end{pmatrix}\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}$$

The motors are controller by two Roboteq SDC2130 motor controllers (see Appendix for details of the motor controllers).

3 DoF Waist

The waist of the robot is actuated by four linear actuators, with the top and bottom plates connected by springs, allowing the upper body to move in three axes: rotation about x and y, and translation in z:



The actuators are controlled by two Roboclaw motor controllers (see Appendix for details of the motor controllers) and the feedback is monitored via an Arduino MEGA 2560.

3 DoF Arm

Mounted on the top plate of the waist, the 3 DoF arm is composed of three servos, controlled by a Pololu Micro Maestro servo controller (see Appendix for details of the controller):

<Insert arm photo here>

2 DoF Head

A pan-tilt mechanism, mounted on the head and also controlled by the Pololu Micro Maestro, serves as a mount point for Jeeves' head:

< insert pan/tilt servo assembly photo here>

Sensors

Encoders

Each wheel is connected via gear train to a Yumo A6A2-CW5C optical encoder, providing odometry at 400 pulses per rev (see Appendix for details):

<insert encoder photo here>

Odometry model.

The pose of the robot is continually estimated from wheel velocities using a simple deterministic model:

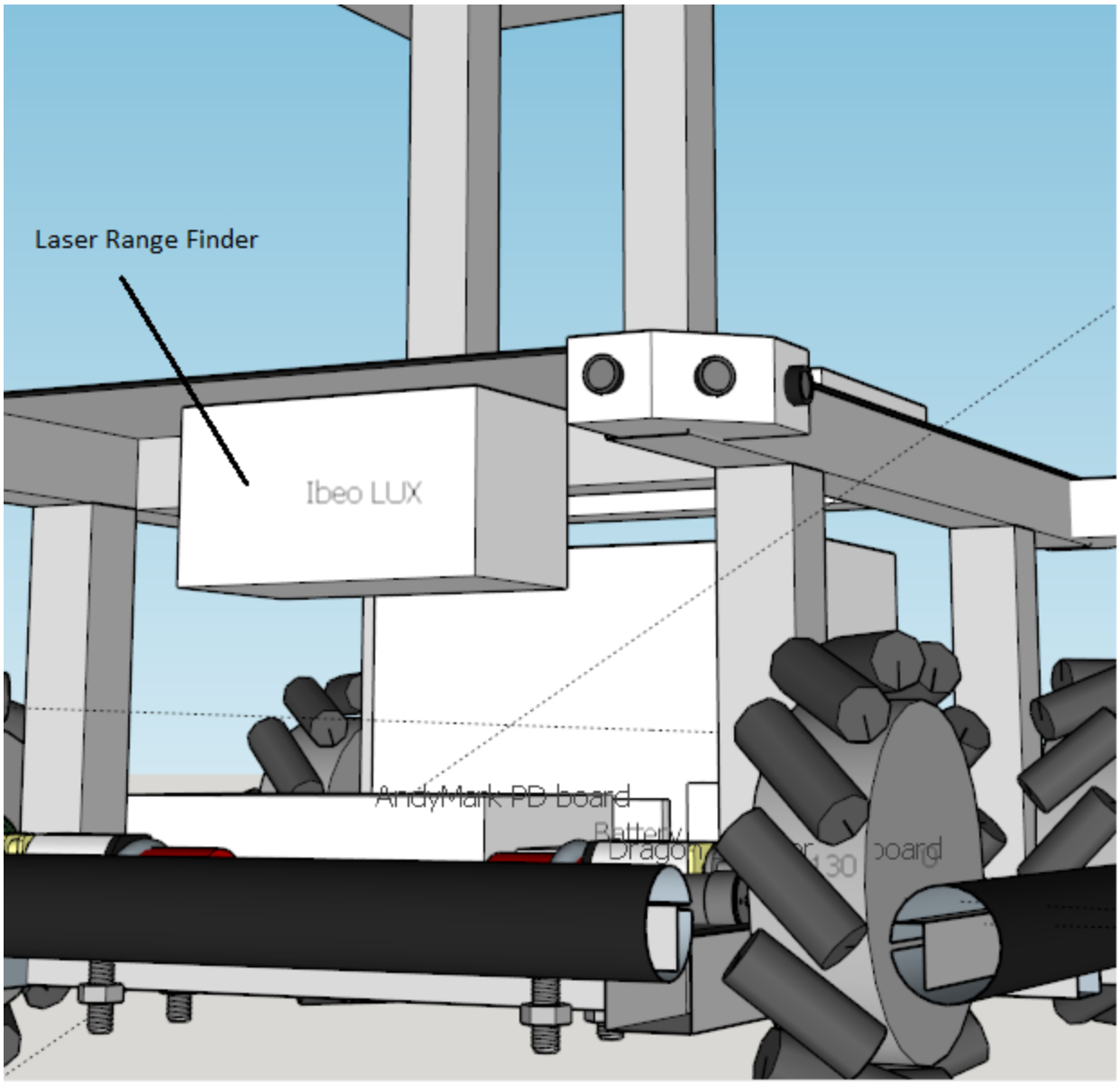
$$\Delta x = (v_x \cos\theta_{t-1} - v_y \sin\theta_{t-1})\Delta t$$
$$\Delta y = (v_x \sin\theta_{t-1} + v_y \cos\theta_{t-1})\Delta t$$
$$\Delta\theta = \dot{\theta}\Delta t$$
$$x_t = x_{t-1} + \Delta x$$
$$y_t = y_{t-1} + \Delta y$$
$$\theta_t = \theta_{t-1} + \Delta\theta$$

where

θ = robot heading in the odometry frame (angle w.r.t the x-axis),
x, y = position in the odometry frame
$\theta(dot)$ = angular rate, calculated from wheel velocities
v_x, v_y = linear rates, calculated from wheel velocities
Δt = time step (20 ms)

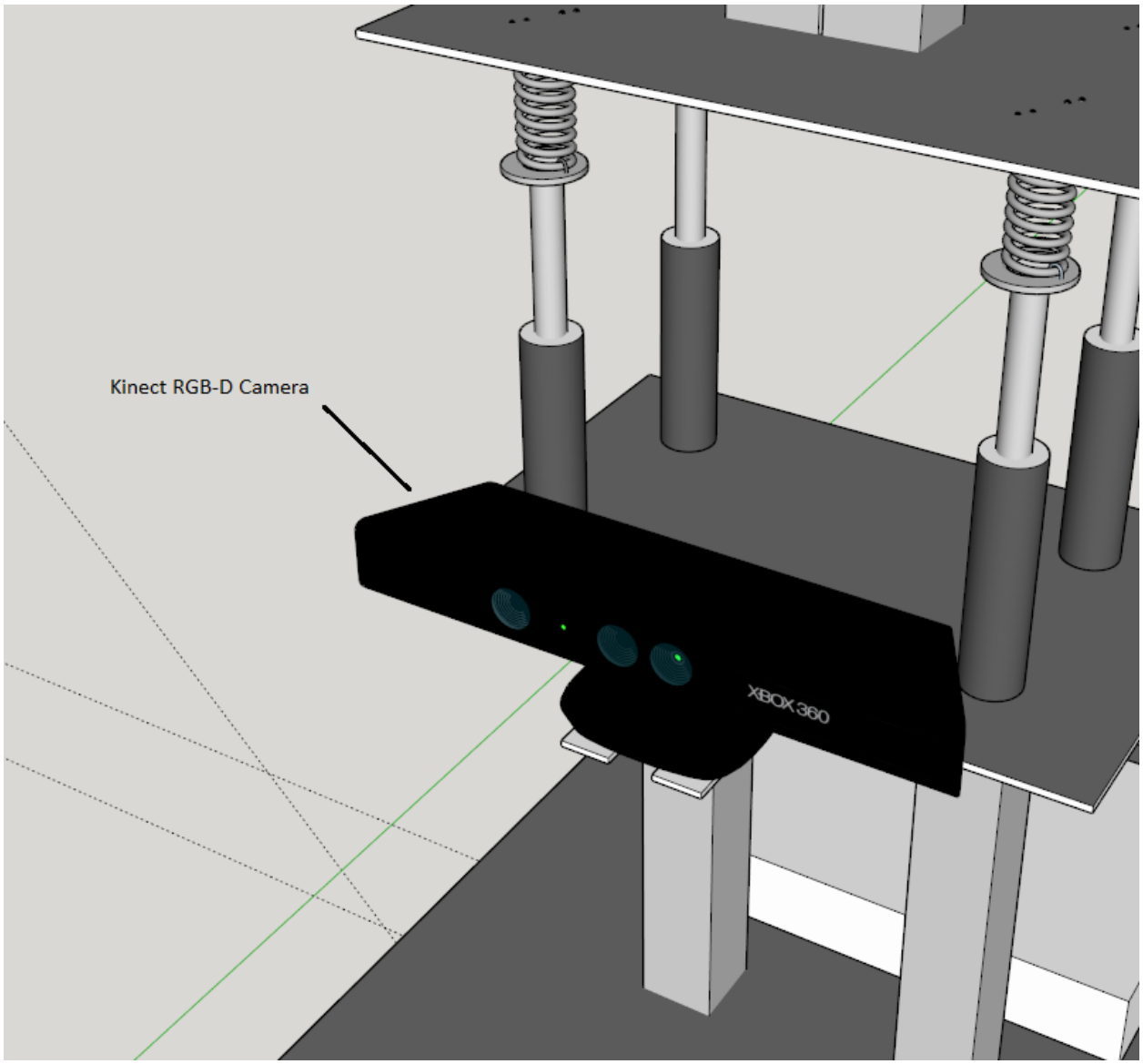
Laser Range Finder

Jeeves has an Ibeo LUX automotive-grade laser range finder, with a 90-degree field-of-view and up to 200m range, measuring at rates up to 50 Hz:



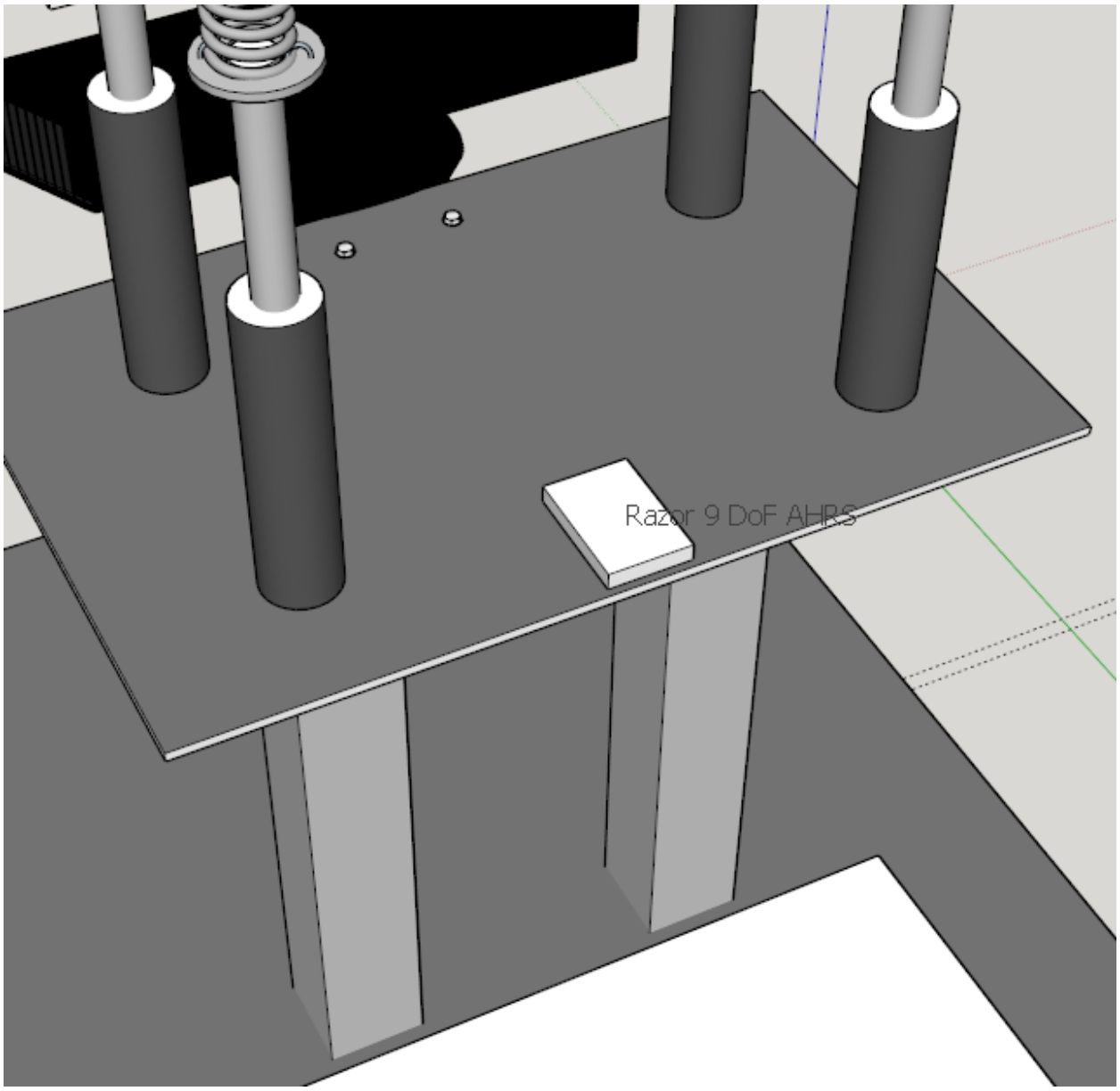
Kinect

Jeeves has Kinect RGB-D camera, connected via USB 2.0 to the main computer. The Kinect is useful for human-robot Interaction and obstacle detection (see Appendix for details):



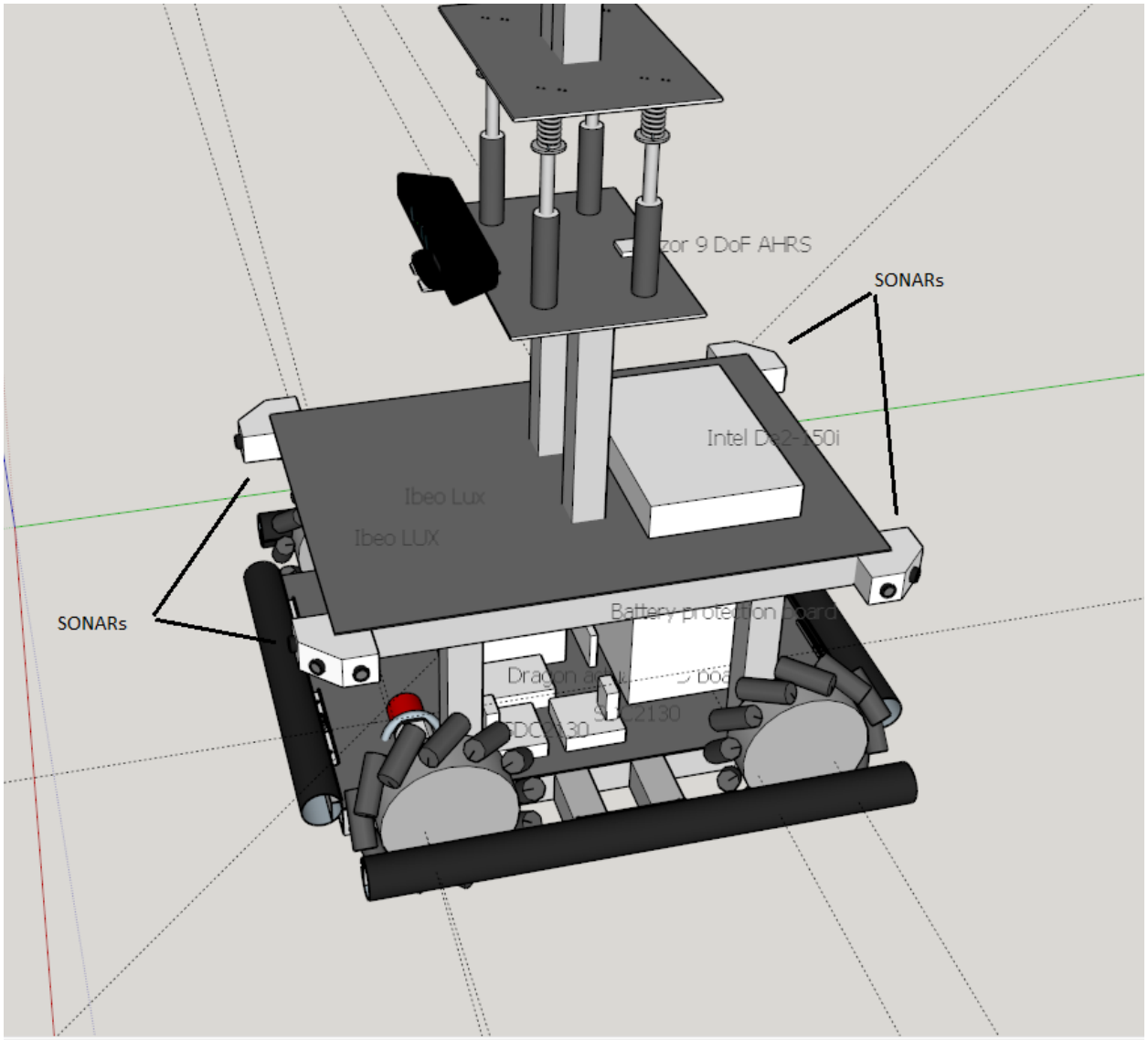
IMU

Jeeves has a 9 DoF IMU: 3-axis accelerometer, 3-axis rate gyro, 3-axis magnetometer:



SONAR

Jeeves has 12 Maxbotics SONARs, for a 360-degree field-of-view. The SONARs are connected to an Arduino microcontroller, which is connected to the main comuter via USB 2.0:



E-stop, Main Breaker

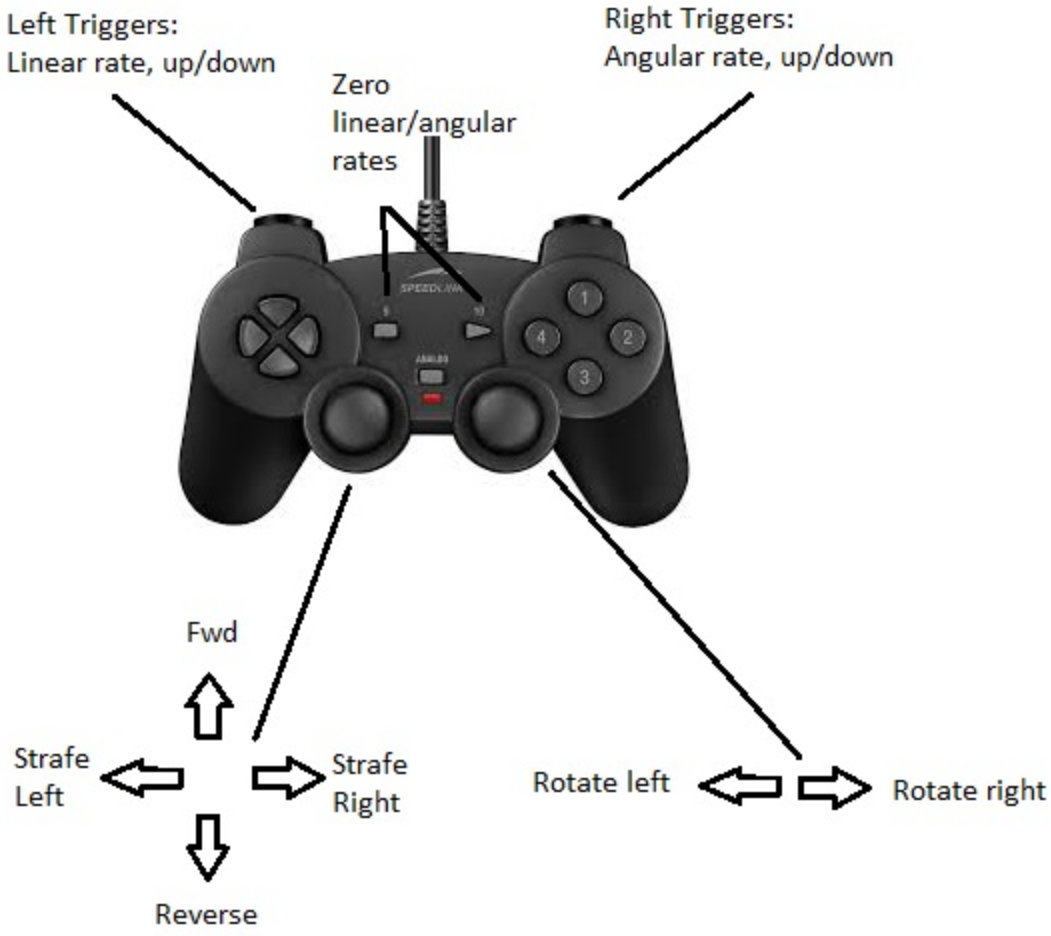
Jeeves is equipped with an emergency stop button that cuts off power to all actuators, while leaving all computers active:

<insert photo of e-stop here>

Jeeves also has a master power breaker, which completely cuts off all power to the platform, whether it be from battery or “shore power”.

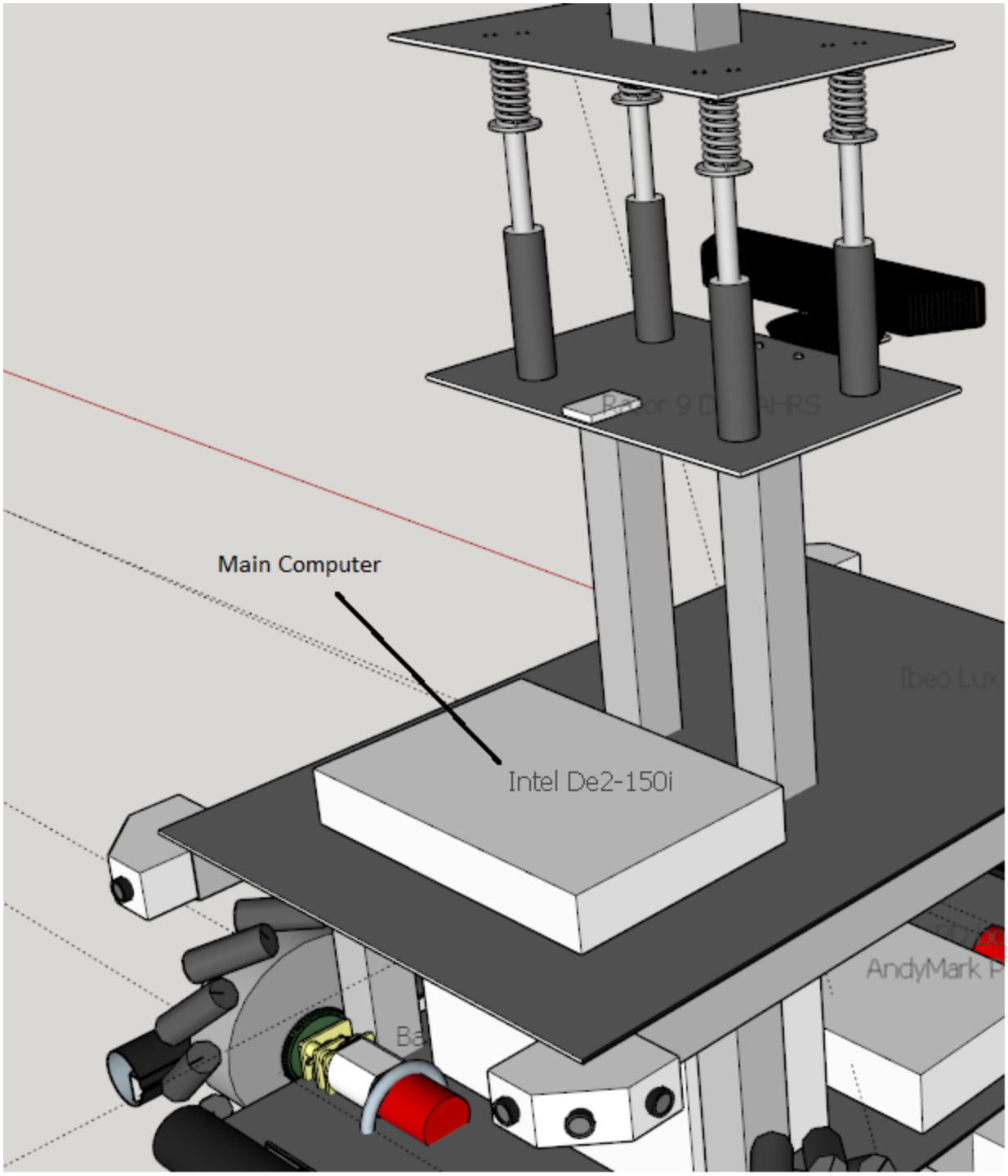
Gamepad

Jeeves is equipped with a standard gamepad for manual operation of the base, and for switching Jeeves’ system software between its various modes:



Main Computer

Jeeves' main computer is an Intel DE2i-150 FPGA Development Kit, with an Atom (Cedarview) processor and an Altera Cyclone IV FPGA connected via PCI-e and Ethernet. The Atom processor runs Xubuntu Linux:



Software Architecture (ROS)

Subsystem Details

Arm Control

Waist Control

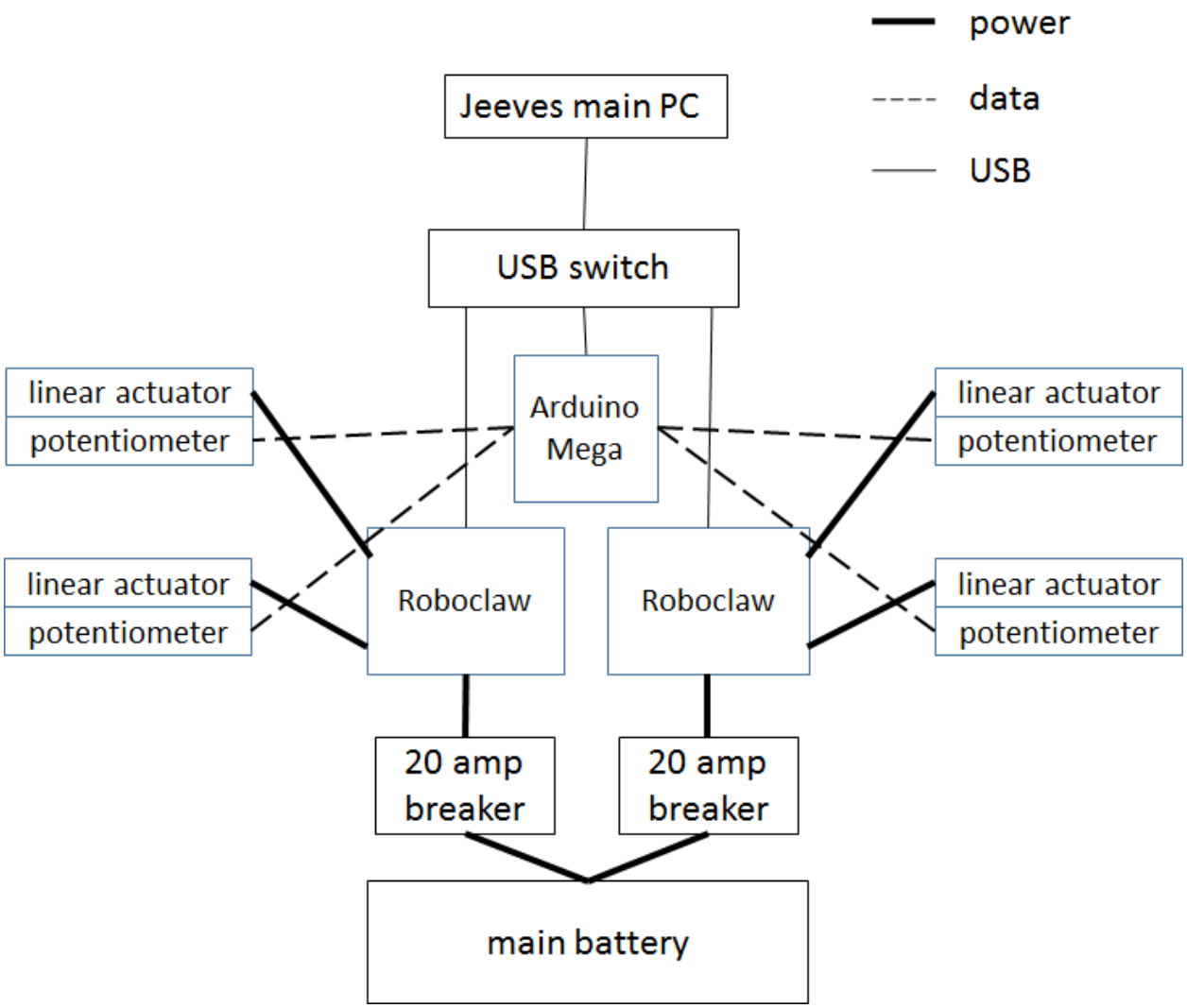
The upper part of jeeve's waist can be moved by 4 linear actuators (type: Frigelli Automations FA-PO-35-12-4" (5:1)). The maximum range of the actuators is 4" (= 101.6mm) and the travel limits are enforced by limit switches. To operate in a secure range, the maximum range is set by the code to 90 mm. The actuators are connected via springs to the upper plate of the waist, which allows the waist to bend.

The bending is implemented as follows:

- forward bending (rotation y positive):
    - rear actuators moving up
    - front actuators moving down
- backward bending (rotation y negative):
    - rear actuators moving down
    - front actuators moving up
- sidewards bending right (rotation x positive):
    - right actuators moving down
    - left actuators moving up
- sidewards bending left (rotation x negative):
    - right actuators moving up
    - left actuators moving down

Forward/backward bending can also be combined with sidewards bending at the same time. There are different ways to initialize a bending poses and linear actuator movements. In general the linear actuators can be manually controlled with the direction pad of the game controller, direct commands can be set in the software and the waist is bending according the velocity of the base, if Jeeves is moving. A more detailed description will be given in the Software API section.

The linear actuators are controlled by two Roboclaws, one Roboclaw can control two motors and all four feedback readings from the potentiometers, which are included in the linear actuators, are monitored by one Arduino MEGA 2560. The Roboclaws are individually powered via two 20 amp breakers. The following block diagram shows the schematic setup of the waist with power and data flow:



The Arduino reads the potentiometer values with a loop rate of 5Hz. All other loops are slaved by this loop. The voltage of the potentiometer varies between 0.59 V at the lowest point and 4.75 V when the linear actuators are completely extended.

There are udev entries for both Roboclaws and the Arduino. For detailed information how to create and modify udev entries, see “Adding a new USB device for Jeeves (udev configuration). The udev entries are:

- Roboclaw left: “waist_actuator_control_left”
- Roboclaw right: “waist_actuator_control_right”
- Arduino board:“waist_feedback”

The Roboclaws do not have an internal serial number or something similar that could be used to explicitly allocate a Roboclaw to a udev entry. This is why the Roboclaws are allocated via the USB port they are connected to. The according USB ports are labeled at the USB switch and may not be changed.
The Roboclaws are set to USB mode, which allows the instant appearance as a USB device at any time. If the Roboclaws are not recognized by the main PC please first check if the USB cables are properly connected to the correct port at the switch, then check if the Roboclaws are set to USB mode. How to check and change the mode, please see Roboclaw manual in the appendix.

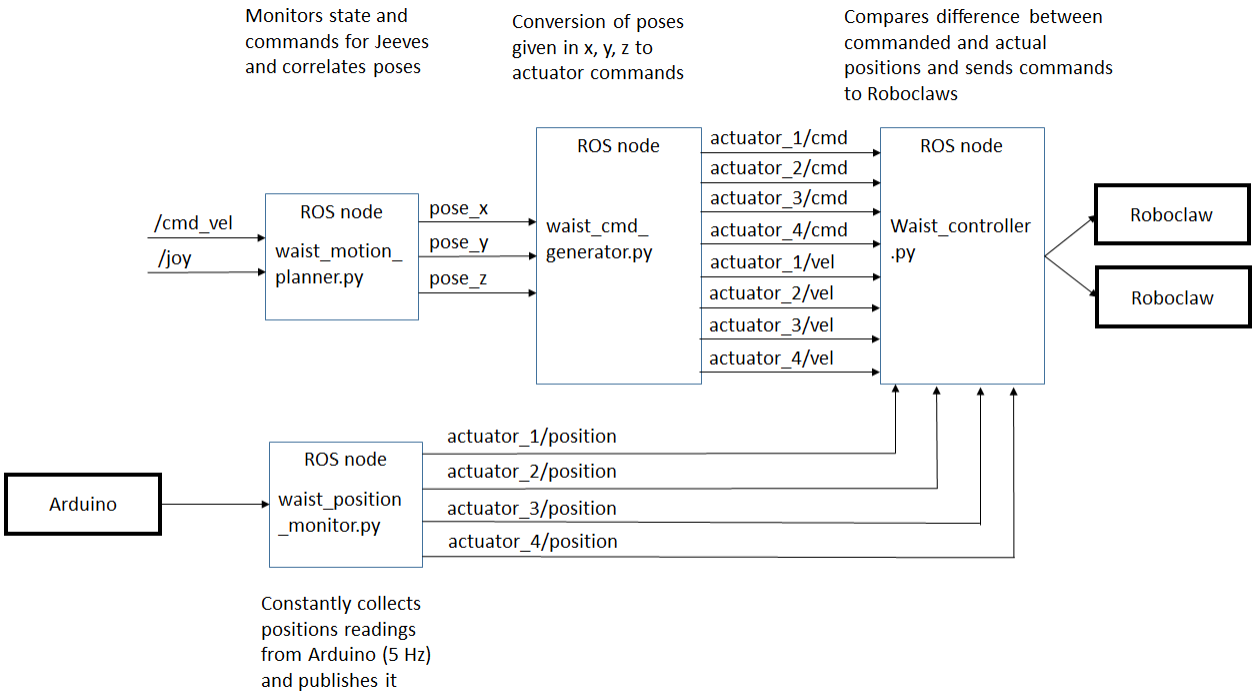
- “See data sheet for linear actuators in appendix.”
- “See data sheet for Roboclaws in appendix.”
- “See wiring diagram in appendix.”

Specifications

- Linear range, z-axis: 90mm programmed (10” = 101.6mm max)
- Max linear rate, z-axis: 50mm/s
- Angular range, x-axis: ∓ 18°
- Angular rang, y-axis: ∓ 28°
- Max angular rate, x-axis, y-axis:
- Distance between actuators, x-axis: 165mm
- Distance between actuators, y-axis: 265mm
- Stall current: ~ 50 A

Software API

The ROS package “waist_control” contains all code to control the waist of Jeeves. It is temporarly located at: “~/mariob/waist_ws/src” and needs to be moved and included in the main source code tree. This package enables the user to control the waist manually with the direction pad of the game controller and automatically triggers waist bending according to base movements. The general software architecture is shown in the following graph:



Input

Inputs are coming from the Arduino as values between 0 and 1023, from the /cmd_vel topic as geometry_msgs.msg Twist and from the /joy topic as sensor_msgs.msg Joy.

Output

Outputs are written to the two Roboclaws via the predefined serial ports and the roboclaw_waist node.

Topics

pose_x, pose_y, pose_z (waist_control.msg/CommandPosition)
angles (pose_x and pose_y) in degree in height (pose_z) in mm

actuator_(1-4)/cmd (waist_control.msg/CommandPosition)
commanded positions for linear actuators in a range from 0 to 1023

actuator_(1-4)/vel (waist_control.msg/CommandPosition)
commanded velocities for linear actuators in a range from 0 to 64

actuator_(1-4)/position (waist_control.msg/FeedbackPosition)
potentiometer values from linear actuators in a range from 0 to 1023

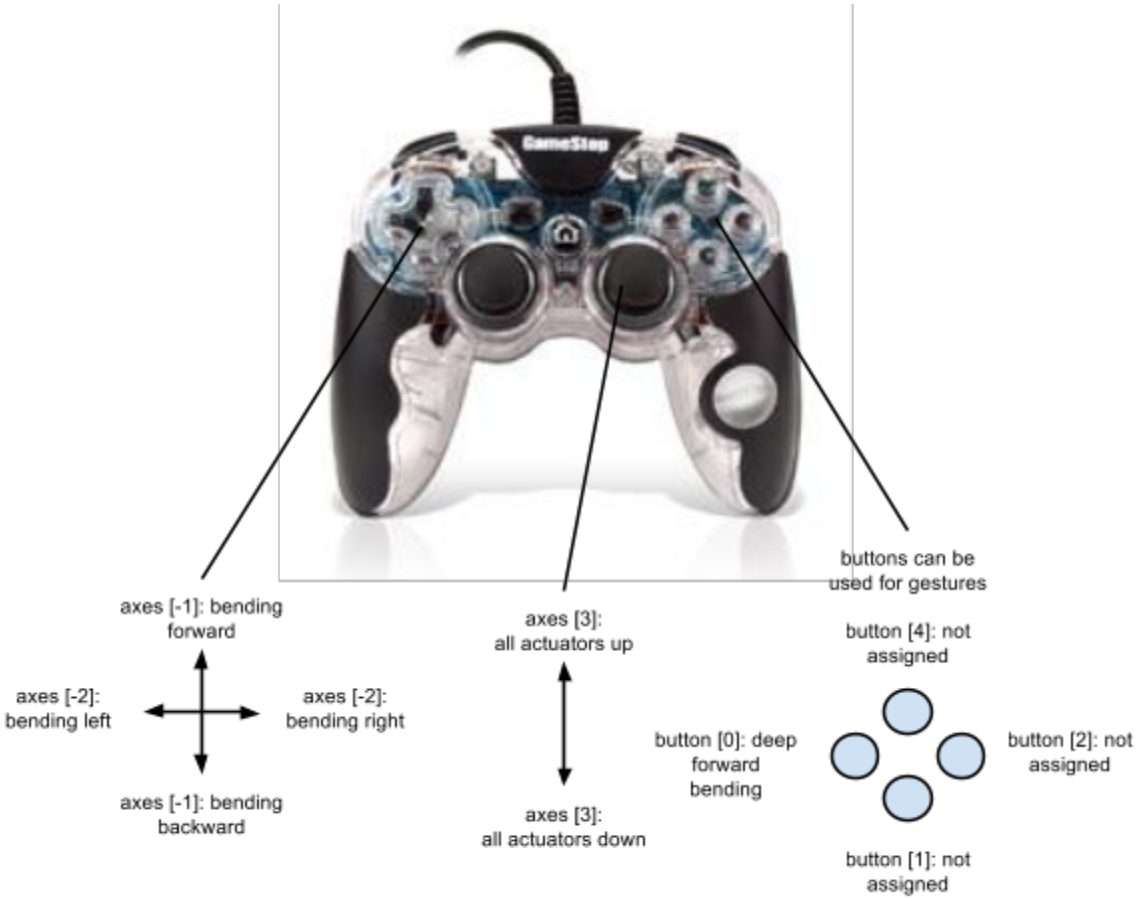
Nodes

The following chapter briefly describes the four nodes that are used to control the waist. More detailed information about how the nodes are working can be found in the comments of the actual code, that is written in Python. All code is attached in the apendix.

waist_motion_planner_joy.py

subscribes to: /cmd_vel and /joy
publishes to: pose_x, pose_y and pose_z

This node receives the inputs from the joystick and the base movement. ROS has a package called “joy” that includes a node called “joy_node” that enables the use of the game pad with Jeeves. The following picture shows the Gamestop game controller with the axis and button names and the associated control functions for the waist. For further information about the /joy topic execute “rostopic echo /joy” in a terminal.



The direction pad can be used to manually bend the waist and the right analog stick can be used to move the whole waist upwards and downwards. There is also one forward bending pose assigned to the button with the pink square (button[0]). This manual control

can only be used as long as no velocity inputs are coming from the base. These velocity inputs (/cmd_vel) have a higher priority. The waist is reacting to the velocity inputs to simulate an inertness of the upper body. When Jeeves is moving forwards, the waist is bending backwards, when Jeeves is moving to the right, the waist is bending to the left, etc. How to move and control the base, please see the specific section.

waist_cmd_generator.py

subscribes to: pose_x, pose_y, pose_z
publishes to: actuator_(1-4)/cmd and actuator_(1-4)/vel

This node converts the incoming poses to commands that can be matched with the potentiometer readings. The code example shows the conversion for the x and y angles to the command for one actuator.

```
# this radiant value is a float variable
self.cmd_rad_x = math.radians(self.cmd_x_degree)
self.cmd_rad_y = math.radians(self.cmd_y_degree)
# calculate the individual actuator command positions in mm
self.cmd_1_float = self.cmd_z_mm - (self.width/2) * math.tan(self.cmd_rad_x) - (self.depth/2) *
math.tan(self.cmd_rad_y)

# convert mm values (0 - 90) to potentiometer range (100 - 900)
self.cmd_1_conv_float = self.cmd_1_float * self.conv_factor_mm_to_pot
self.cmd_1_conv = int(self.cmd_1_conv_float)
self.cmd_1 = self.cmd_1_conv + self.min_pos  # final cmd height converted to the potentiometer values
```

waist_controller.py

subscribes to: actuator_(1-4)/position, actuator_(1-4)/cmd, actuator_(1-4)/vel

This node constantly compares the incoming commanded positions with the actual positions and controls the linear actuators. The position accuracy can be defined with the global variable “range_position”. The higher the velocity, the bigger needs to be the range position in order to control the actuators. The move commands for the linear actuators are executed by the predefined roboclaw function “DriveM1/2”. This function from the class RoboClaw needs one velocity input. 64 stands for stop, values smaller than 64 move the waist upwards and values greater than 64 move the waist downwards with 0 respectively 123 as the highest possible speed. The following code shows the control procedure for one actuator.

```
#control for actuator right back
self.rangeup_1 = self.cmd1 + range_position
self.rangedown_1 = self.cmd1 - range_position
```

```
if self.pos1 < self.rangedown_1:
    print "I should move up"
    self.r1.DriveM1(64 - self.vel1)
elif self.pos1 > self.rangeup_1:
    print "I should move down"
    self.r1.DriveM1(64 + self.vel1)
else:
    print "reached position"
    self.r1.DriveM1(64)
```

Attention: I lost randomly the connection to the Roboclaw for the two left actuators. I could not find a reasons or an actual solution, but constantly resetting/reopening the port helped to reliably control the actuators.

waist_position_monitor.py

publishes to: actuator_(1-4)/position

This node constantly reads the incoming potentiometer values from the Arduino with a loop rate of 5 Hz. It receives a string and cuts the string in 4 numbers and converts it to integers. The .ino file for the Arduino can be found in ~/mariob/ReadAnalogPosition

Launch file

This package can be launched by the launch file “waist_control.launch” in the package …..directory:

“roslaunch waist_control waist_control.launch”

```
mcecsbot@jeeves:~/mariob/waist_ws$ roslaunch waist_control waist_control.launch
... logging to
/home/mcecsbot/.ros/log/33b72052-ea23-11e3-8ea9-00012e2f3cf4/roslaunch-jeeves-79
79.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://jeeves:33341/
SUMMARY
========
PARAMETERS
* /rosdistro
* /rosversion
NODES
/
```

waist_cmd_generator_node (waist_control/waist_cmd_generator.py)
waist_controller_node (waist_control/waist_controller.py)
waist_motion_planner_node (waist_control/waist_motion_planner_joy.py)
waist_position_monitor_node (waist_control/waist_position_monitor.py)
ROS_MASTER_URI=http://localhost:11311
core service [/rosout] found
process[waist_position_monitor_node-1]: started with pid [7997]
process[waist_cmd_generator_node-2]: started with pid [7998]
process[waist_controller_node-3]: started with pid [8001]
process[waist_motion_planner_node-4]: started with pid [8006]

Writing a Jeeves Application

Operating Procedures

- Mode1: Moving the base using the gamepad or Rviz.
- Adding a new USB device to Jeeves (udev configuration).
- Running the obstacle avoidance demo.

Appendix

Additional documentation

- Redmine site
- youtube pages
- Chapter 5 of Omar's thesis
- Sketchup model

Electrical schematics/diagrams

- System wiring diagram.
- Waist control subsystem wiring diagram.

System Configuration Files

Electrical Components

- Main computer.
- Motor controllers.
    - Telemetry script.
        - Modifying the telemetry script.
- AndyMark PD board.
- Battery protection board.

- Ibeo LUX LRF.
- Linear actuators waist.
- Roboclaws (waist).
- Arm/head servos.
- Pololu Micro Maestro
- Actuator (Dragon) power board.

References

[1] Villiers, M. De, and N. S. Tlale. "Development of a Control Model for a Four Wheel Mecanum Vehicle." *Journal of Dynamic Systems, Measurement, and Control* 134.1 (2012): 011007. Print.

[2] Doroftei, Victor Grosu Ioan, and Veaceslav Spinu. "Bioinspiration and Robotics: Walking and Climbing Robots, chapter 29 Omnidirectional Mobile Robot-Design and Implementation." *I-Tech Education and Publishing, Technical University of Iasi, Romania* (2007): 544.