Home Problem 2.2 Humanoid robots

TIF160

Sara Larsson, 981005-2465

October 6, 2021

Contents

1 Hubert kinematics

Following information is taken from home problem 2.1 where the forward kinematics of Hubert was calculated.

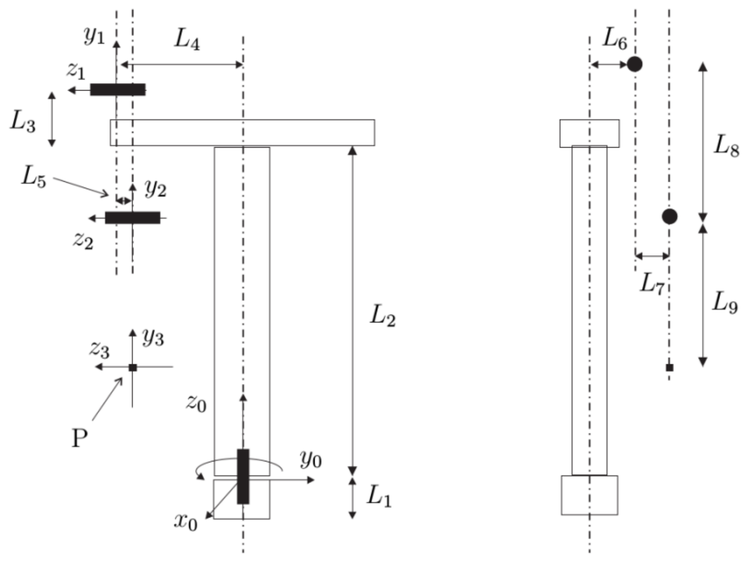

Figure 1: Drawing of Hubert with the positions of the joints, different lengths and the coordinate systems. *Source: Home problem description*

L_1: 0.055	L_4: 0.108	L_7: 0.015
L_2: 0.315	L_5: 0.005	L_8: 0.088
L_3: 0.045	L_6: 0.034	L_9: 0.204

Table 1: Values for lengths in figure 1. *Source: Home problem description*

1.1 Model

The length L_1 is not considered in this part of the home problem since frame 0 is situated on top of L_1.

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_1 & 0 & sin\theta_1 & L_6 cos\theta_1 + L_4 sin\theta_1 \\ sin\theta_1 & 0 & -cos\theta_1 & L_6 sin\theta_1 - L_4 cos\theta_1 \\ 0 & 1 & 0 & L_2 + L_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_2 & -sin\theta_2 & 0 & L_7 cos\theta_2 + L_8 sin\theta_2 \\ sin\theta_2 & cos\theta_2 & 0 & L_7 sin\theta_2 - L_8 cos\theta_2 \\ 0 & 0 & 1 & -L_5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_3 & -sin\theta_3 & 0 & L_9 sin\theta_3 \\ sin\theta_3 & cos\theta_3 & 0 & -L_9 cos\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{pmatrix} \tag{3}$$

1.2 Script

The function-script "forwKinematicsModel.m" for kinematics model: The model uses the derived matrix-multiplication given in eq. 1, 2 and 3. The target point in frame 3 is always located at the origin and is therefore set to [0,0,0,1].

2 Inverse kinematics using LGP

Task: Calculate the inverse kinematics for the humanoid robot shown in figure 1, using Linear Genetic Programming. The best chromosome found in the LGP should give the angle-configurations for the robot to reach a given position.

The code is implemented in Python consisting of 3 classes. *LGP* that inherits from *FitnessFuntion* that inherits from *ForwKinematics*, all classes written and tested in the python-files with the same name.

- LGP takes care of the GA-functions and keeps track of the population.
- FitnessFuntion contains all the calculations to calculate the components for the fitness, the restrictions in space and handles the registers.
- ForwKinematics contains calculation of the forward kinematic model derived in 1.

Most of the calculation are done using Numba, which is a library in python that makes it possible to automatically translate some code to C in order to make significantly faster calculations. Most of the functions used in the library Numpy are supported by Numba. @njit that marks those functions means that all of the code written in the function is non-python and is possible to translate. Therefore the Numba-calculations must be outside the class since it cannot handle the python argument *self.*

2.1 Population

The population is built using a python dictionary. It is initialized by for each individual, creating an array with randomized values representing a chromosome. The length of the chromosomes are randomized with a value divisible by 4 (number of genes that creates an instruction). An instruction consist of an operator, a destination register and two operators. The chromosome for a individual is then put as a value in a individual dictionary with key "chromosome". The Individual-dictionary is then put in the population dictionary with the individual number as key. The fitness is later added in similar way when calculated. An example of the structure is given below with population size 2.

Population:

- 1:
  - "chromosome":[2,5,3,1]
  - "fitness": 3.4
- 2:
  - "chromosome":[3,5,2,4,4,2,1,4]
  - "fitness": 1.3

2.2 Registers

The registers contains of a set of variable registers and constant registers. The 3 first variable registers are used as input, i.e. the target position in (x,y,z). Since there is no dataset given, a random set of points within the robots range (given below) will be sampled. It will help the LGP since the unreachable space is huge!

Range restrictions (approximated):

- Restrictions of angles: $\theta_1 = [0 : \pi]$, $\theta_2 = [0 : \pi]$, $\theta_3 = [0 : \frac{\pi}{2}]$
- Axis:
  - x-min $= -(L_6 + L_8 + L_9)$ when $\theta_1 = \pi$, $\theta_2 = \frac{\pi}{2}$, $\theta_3 = 0$, see second picture in 2.
  - x-max $= \sqrt{(L_4 - L_5)^2 + (L_6 + L_8 + L_9)^2}$, the length of the imaginary line from the origin of frame 0 to the target point P, when $\theta_2 = \frac{\pi}{2}$, $\theta_3 = 0$ and the robot is rotated with θ_0 so that the imaginary line is aligned with x_0, see first picture in 2.
  - y-min $= L_6 - L_7 - L_9$ when $\theta_1 = 0$, $\theta_2 = \pi$, $\theta_3 = \frac{\pi}{2}$, see second picture in 2.
  - y-max $= \sqrt{(L_4 - L_5)^2 + (L_6 + L_8 + L_9)^2}$, the length of the imaginary line from the origin of frame 0 to the target point P, when $\theta_2 = \frac{\pi}{2}$, $\theta_3 = 0$ and the robot is rotated with θ_0 so that the imaginary line is aligned with y_0, see first picture in 2.
  - z-min $= L_2 + L_3 - L_8 - L_9$, all $\theta = 0$. Robot having arm down.
  - z-max $= L_2 + L_3 + L_8 + L_9$ when $\theta_2 = \pi$ and $\theta_3 = 0$. Robot having arm up.
- Sphere: The positions are then also restricted to inner and outer bounds that creates spheres, see figure 3.
  - Outer bound: The positions are then also restricted to the sphere that a outstretched arm is creating when rotating theta 1 and theta 2. The center of the sphere is therefore $center = (0, 0, L_2 + L_3)$. The same reasoning as for y-max and x-max is done here for getting the max radius of the sphere. $radius_{outer} = \sqrt{(L_4 - L_5)^2 + (L_6 + L_8 + L_9)^2}$.
  - Inner bound: The inner bound has two spheres that rotates depending on the point's position in z-axis. The limit is positioned where joint 3 is located when $\theta_2 = 0$, mainly $limit = L_2 + L_3 - L_8$.
    * Upper: The shortest distance towards the center is when $\theta_3 = \frac{\pi}{2}$. The radius of the sphere is therefore $radius_{innerUpper} = \sqrt{\sqrt{L_8^2 + (L_7 + L_9)^2}^2 + (L_4 + L_5)^2}$ which is using two Pythagoras calculations. The center of the sphere is located in the same position as for the outer sphere.
    * Lower: The shortest distance towards the center is when $\theta_2 = 0$. The radius of the sphere is therefore $radius_{innerUpper} = \sqrt{L_9^2 + (L_4 - L_5)^2}$. The center of the sphere is located at joint three $center_{lower} = (0, 0, L_2 + L_3 - L_8)$.

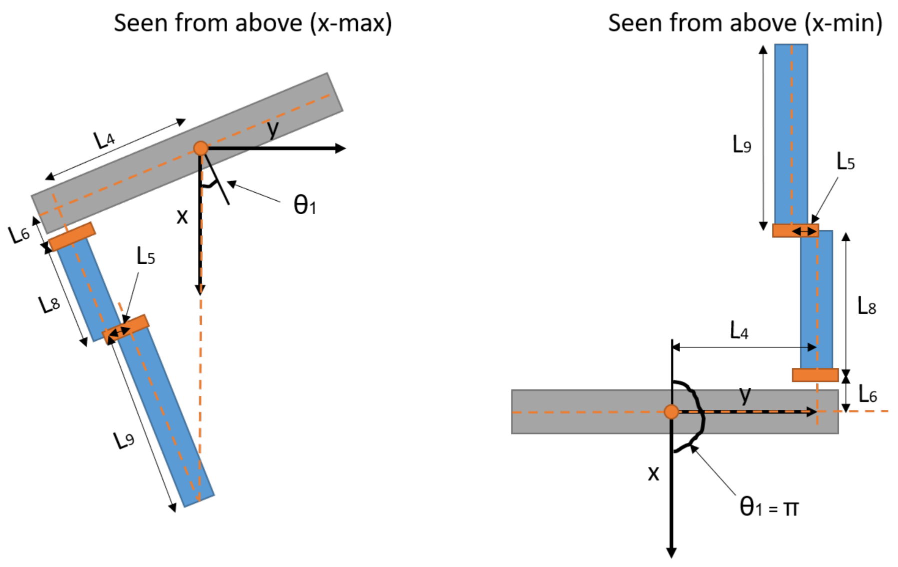

Figure 2: Min and max in x-direction

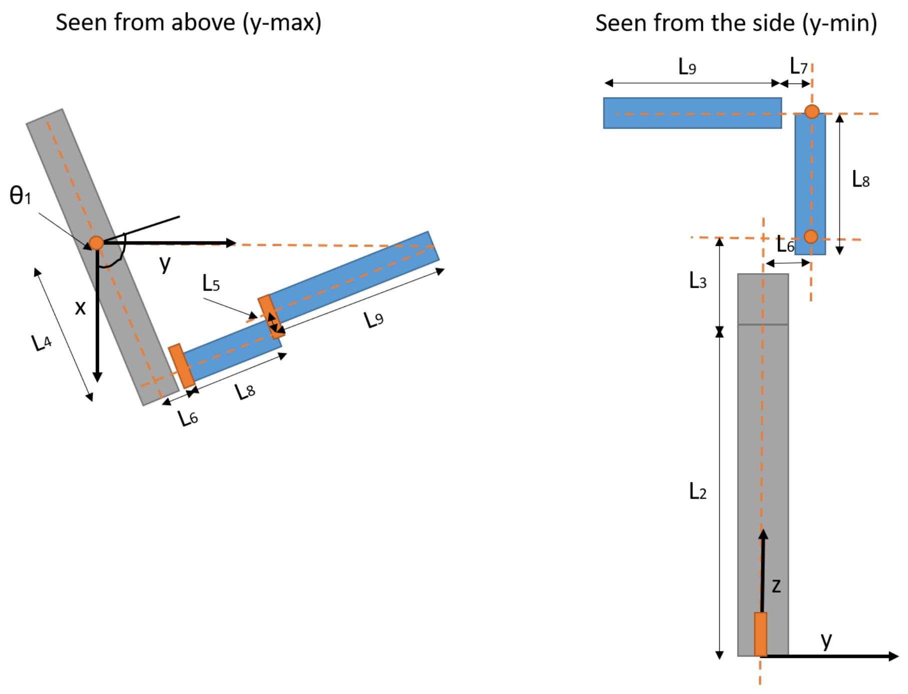

Figure 3: Min and max in y-direction

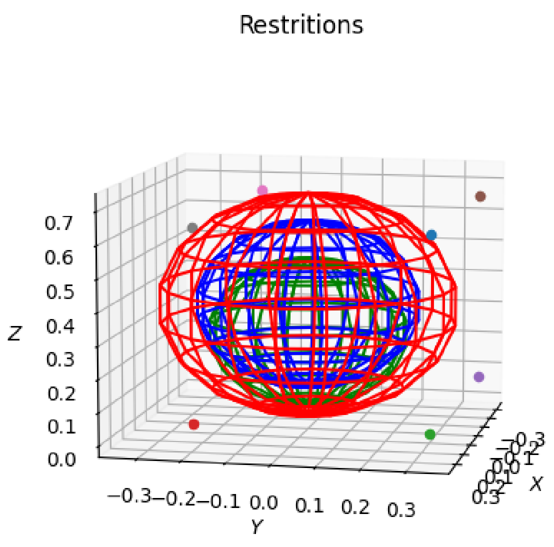

Figure 4: The colored dots are the corner of the rectangle that max and min values of all axis creates. The red sphere is the outer bound with the sphere center located at the torso in x- and y-axis, and joint two in z-axis. The blue is the upper inner bound with same center point as the outer bound. The green is the lower inner bound with center point located at the same x- and y-coordinates but at joint three in z-axis.

2.3 Fitness function

To determine the fitness of an individual, the chromosome is first decoded to get the predicted angles using the position of all data points. Then the forward kinematics derived in home problem 1 is used to calculate the errors which the LGP should minimize. The fitness of an individual is therefore determined by taking $\frac{1}{error}$. The error and decoding is described below.

There is as well a max length given for the chromosomes to not get too long chromosomes during training. If the chromosome is longer than the max length, then the chromosome will be given a penalty by having it's fitness multiplied with a penalty term < 0.

2.3.1 Decode chromosome

The calculations are using Numba (description in the beginning of chapter 2) in order to get faster calculations.

To decode the chromosome, the instructions are iterated by taking every quadruple of genes. The first gene is the operator and are: [+, -, *, /, sin, cos, arc-sin, arc-cos] since the trigonometric operators might be needed to predict angles. The second gene is the destination register (any of the variable registers) and the last two, the operands which can use all registers.

Division by 0 is eliminated setting the value of the destination register to a huge value directly. The operands are also restricted between -1000000 and 1000000 for the registers to not to go towards infinity or negative infinity. When doing the trigonometric instructions, only operand 1 is used. Another restriction is since arc-sin and arc-cos only takes input between -1 and 1, the operand is therefore used by taking modulus 2 minus 1.

The first three variable registers are then the output and gives the predicted angle configurations. Since the angles of the motors are restricted (see 2.2), the angle is restricted by taking modulus of max angle for each theta.

2.3.2 Calculate error

The error is determined by taking the Euclidean distance of the coordinates of the actual points gotten from the three first variable registers and the predicted points. The predicted points are derived by using the forward kinematics from 1 which has been translated from Matlab-code to Python-code as a class with Numba-functions. If the distance is 0, then the distance is set to a extremely small number to eliminate the risk of division by 0. To calculate a final error, a rms-error is calculated from the collection of errors.

2.4 LGP methods

The new population is created in pairs, selected by tournament selection and modified by crossover and mutation. Elitism is then applied after the new population is created.

- **Tournament selection:**N_T of randomized individuals are sorted by the highest fitness in an array. The next individual in the array will then be selected by a probability P_T. If not, then the second individual will be chosen with the same probability P_T, and so on. The worst individual, i.e. the last one in the array will be chosen if none of the other individuals are chosen.
- **Two point crossover:** Two random points are selected on each chromosome of two individuals. The chromosomes are then divided into three parts by the points and the middle part are swapped, which creates two new individuals. The points are randomized that it does not destroy any instructions (every 4th gene). The crossover happens with a probability P_C.
- **Mutation:** Mutation of the chromosomes happens by replacing a gene in the chromosome with a new randomized gene with a probability of

P_M. The new gene must be of the same type as the old gene (variable register/register/operator).

- **Elitism:** Replaces the first individual in the new population by the best individual in the previous population.

2.5 To run the code

To run the programs, Python and all the used libraries are required. There are 4 python files "LPG.py", "fitnessFunction.py", "forwKinematicsModel.py" and "plotResult.py".

To run the training, run the file "LPG.py". All settings are determined in the function called *main()*.

There are 4 matlab files for the test program. "analyze_result.m", "DecodeChromosome.m", "forwKinematicsModel.m" and "getPositions.m". They are all translated from python code. To run the test script in Matlab, use "analyze_result.m". Set the test nr, folder and number of datapoints you want to test with for the chromosome.

The best chromosome gotten is test 6 that is located in the folder "result". Approximately 70% of the time, the points get errors under 0.1, see figure 5. As seen in figure 6, the robot cannot reach certain positions and that the datapoints might be outside the robot's reach since the restrictions are approximated.

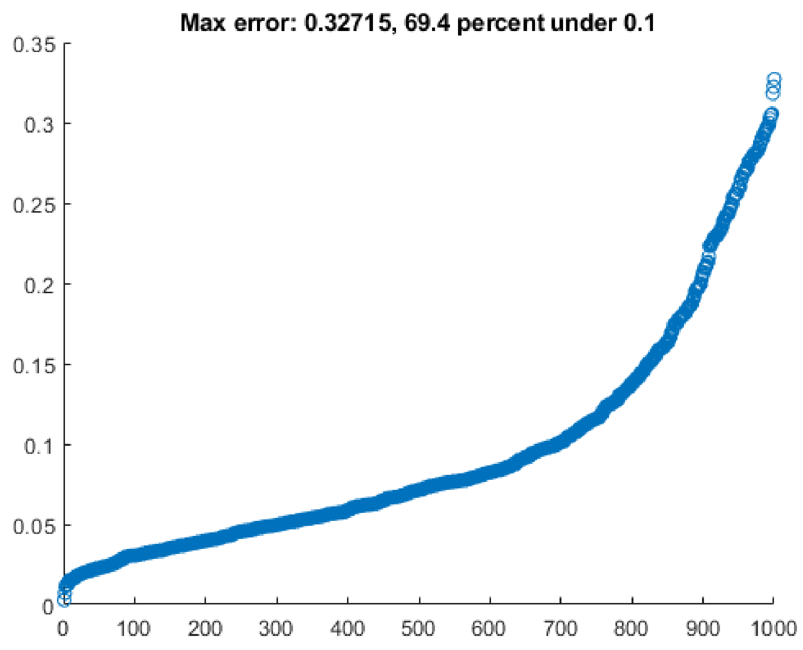

Figure 5: Error of all the datapoints.

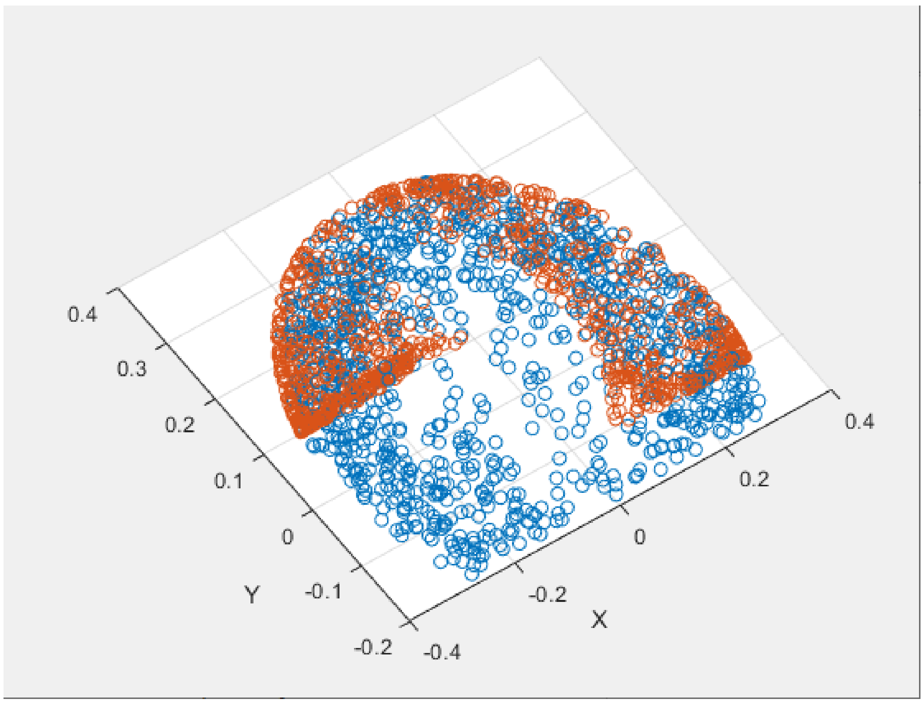

Figure 6: All datapoints (blue) and predicted positions (orange) plotted in 3D-space.