University of Southern Denmark

Project in Advanced Robotics - Spring 2020

Obstacle Avoidance with Dynamic Movement Primitives

Emil Vincent Ancker
09/06-1997
emanc16@student.sdu.dk

Mikkel Larsen
08/08-1996
milar16@student.sdu.dk

Mathias Emil Slettemark-Nielsen
23/02-1995
masle16@student.sdu.dk

Bjarke Engsig Larsen
13/07-1995
blars16@student.sdu.dk

Group number: 8

Supervisor: Iñigo Iturrate San Juan

Project period: 2nd April 2020 - 29th May 2020

Preface

The report is part of a 5 ECTS course, Project in Advanced Robotics at the Technical Faculty, University of Southern Denmark.

The contributions of the individual authors of this project is denoted after each title or subtitle indicating that the following text until a new author name is presented (below a title) is written by that specific author.

An example is shown here:

Title Used for Example

`Author name`

This is a block of text written by "Author name".

Overview of Contributions

The contributions of each member with respect to implementations are stated here.

Rotational DMP	Bjarke Larsen
Position DMP	Emil & Mikkel with equal effort. (with template from Iñigo).
End-effector obstacle avoidance	Emil & Mikkel with equal effort.
Link collision avoidance	Emil Vincent Ancker
3D-3D Pose estimation	Mikkel Larsen
DenseFusion	Mathias Emil Slettemark-Nielsen

Contents

1 Introduction

In this project the framework of dynamic movement primitives will briefly be explored for both position and rotation, before moving into obstacle avoidance for a 7-DOF robot manipulator.

End-effector obstacle avoidance will be incorporated into the framework of dynamic movement primitives, where the usage of repulsive potential fields will be explored. Afterwards, avoidance of collision between the links of the robot and obstacles will be considered and incorporated into inverse kinematics. To achieve a feasible inverse kinematics solution for link collision avoidance it will briefly be looked at how the performance when operating close to singularities can be improved to avoid infeasible joint velocities.

The focus of this project is not to tune the avoidance method into perfection, but more simply to explore the possibilities of incorporating potential fields into the dynamic movement primitive framework and explore link collision avoidance.

Finally, 6D pose estimation will be described and implemented since the avoidance methods assumes the pose of an obstacle is known. The 6D pose estimation can be implemented to pose estimate obstacles and thus deliver a pose that can be used by the obstacle avoidance methods. For 6D pose estimation a technique based on neural networks will be compared to a more traditional 3D-3D pose estimation method.

2 Dynamic Movement Primitives

Bjarke Larsen

The following section describes Dynamic Movement Primitives (DMP).

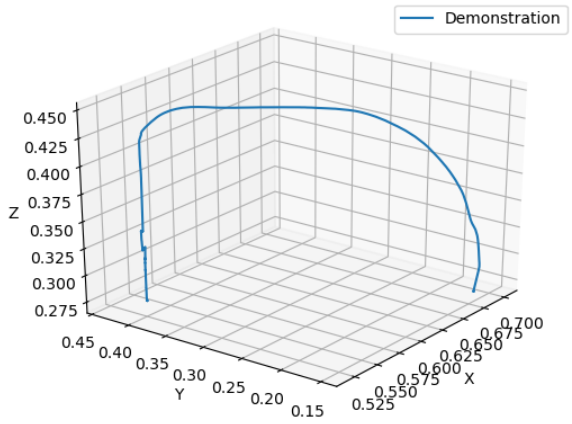


Figure 1: Trajectory demonstration, which is handed to the DMPs.

DMPs provide the option to fit a function to a discrete recording of a trajectory. In Figure 1 a recorded trajectory is shown, which the end effector should follow from the initial position to the goal position. The idea of DMP's is to generalize learning from demonstration, in a way that allows for time scaling and dynamic modification of the trajectory. By recording the trajectory as coordinates for the end effector in 3D space instead of recording the joint values, it allows for modification of the path based on the environment. This could be modification of the goal coordinates or dynamic avoidance of obstacles, of which avoidance of a moving obstacle will be explored in section 3.

The idea of Dynamic Movement Primitives (DMP) is to use an analytically well-understood dynamical system that will converge towards a specific goal point [1] and adding a nonlinear forcing term making the manipulator achieve a desired behavior. A spring-damper system with a forcing term is used:

$$\tau \dot{z} = \alpha_z(\beta_z(g - x) - z) + f, \tag{1}$$

$$\tau \dot{x} = z, \tag{2}$$

The above system in Equation 1 is called the transformation system [1], where τ is a time constant, α_z and β_z are constants used to adjust stiffness and damping. From [1] choosing $\beta_z = \frac{\alpha_z}{4}$ the system is made critically damped, giving the desired features of a quick response without overshoot as x converges to g. The term f is a forcing term. With f being nonlinear, this enables a more useful behavior of the point attractor than a straight

line. The forcing term f is defined by,

$$f(\chi) = \frac{\sum_{i=1}^{N} \psi_i(\chi) w_i}{\sum_{i=1}^{N} \psi_i(\chi)} \chi (g - x_0) \tag{3}$$

Where χ is a phase variable used to make the system time independent. The term $g - x_0$ is a useful scaling when the distance between the goal and initial configuration is altered. Finally by multiplying with χ it is ensured that the forcing term lets x converge to g since the nonlinearity will disappear as χ goes to 0. The nonlinear forcing term is built from a number of weighted basis functions. These basis functions essentially resemble gaussians with different means and variances. The forcing term is then built from a weighted sum of the basis functions, to imitate the recorded path. The individual basis functions $\psi_i(\chi)$ are defined by,

$$\psi_i(\chi) = \exp\left(-\frac{1}{2\sigma_i^2}(\chi - c_i)^2\right) \tag{4}$$

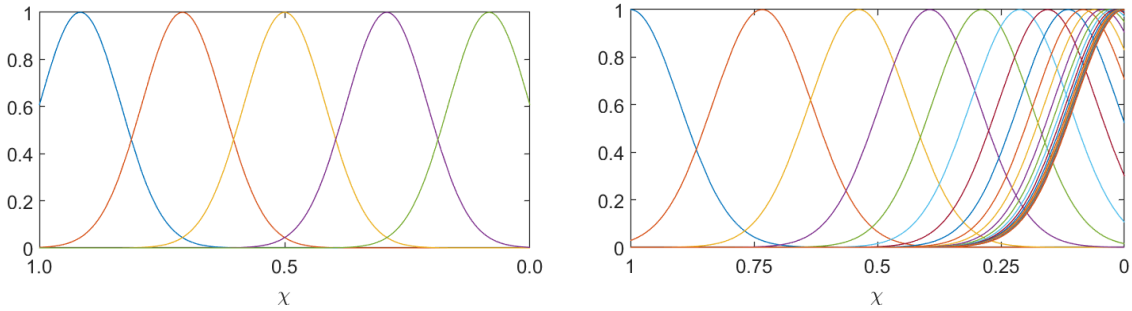
Where σ_i determines the width of the basis function and c_i defines the center of the basis function. Choosing the centers for the basis functions is a matter of design. One option being to place the centers evenly along as the phase goes to zero, while another option could be to cluster the basis functions closer together based on the complexity of the trajectory for a better result with fewer basis functions. The phase variable, χ, is used to avoid dependency on time and thus enabling time scaling. The phase variable is defined by the following first-order linear dynamic system, which is called the canonical system,

$$\tau \dot{\chi} = -\alpha_\chi \chi \tag{5}$$

Since this canonical system decays to zero exponentially the centers of the basis functions are placed as follows:

$$c_i = \exp\left(-\alpha_c \cdot \frac{i}{N}\right) \tag{6}$$

Where N is the total amount of basis functions and α_c is a positive constant. In Figure 2 an illustration of the basis functions is shown with linear spacing and with the exponential centers resulting in an even spacing given the choice of canonical system.



(a) Centers placed linearly as a function of indice.

(b) Centers of basis function placed as an exponential function of indices.

Figure 2: Visualization of the two considered ways of distributing basis function centers.

To make the DMP follow the demonstration shown in Figure 1 the weights w_i in the nonlinear forcing term in Equation 3 needs to be adjusted. Learning these weights can be accomplished from a single demonstration using linear regression. From the transformation system in Equation 1, the following can be seen [1]:

$$\tau \dot{z} - \alpha_z(\beta_z(g - x) - z) = f \tag{7}$$

By inserting the known information from the demonstration in the left side,

$$\tau^2 \ddot{x}_{demo} - \alpha_z(\beta_z(g - x_{demo}) - \tau \dot{x}_{demo}) = f_{target} \tag{8}$$

The objective is now to determine the weights, w_i, that makes the trajectory f come as close to f_{target} as possible. This is now written as a system of linear equations. First the matrix $\mathbf{\Psi}$ is defined, which simply contains the N basis functions along the rows and has K rows, where K is the number of samples in the demonstration. Along with the matrix of basis functions a vector of all the weights is also defined,

$$\underset{(K\times N)}{\mathbf{\Psi}} = \begin{bmatrix} \psi_0 & \psi_1 & \dots & \psi_N \\ \psi_0 & \psi_1 & \dots & \psi_N \\ \vdots & \vdots & \vdots & \vdots \\ \psi_0 & \psi_1 & \dots & \psi_N \end{bmatrix}, \quad \underset{(N\times 1)}{\boldsymbol{W}} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{bmatrix} \tag{9}$$

Finally, Equation 8 is put on matrix form,

$$\underset{(K\times 1)}{\boldsymbol{F}} = \begin{bmatrix} f_{target,0} \\ f_{target,1} \\ \vdots \\ f_{target,K} \end{bmatrix} \tag{10}$$

The system of linear equations to be solved using least squares is then,

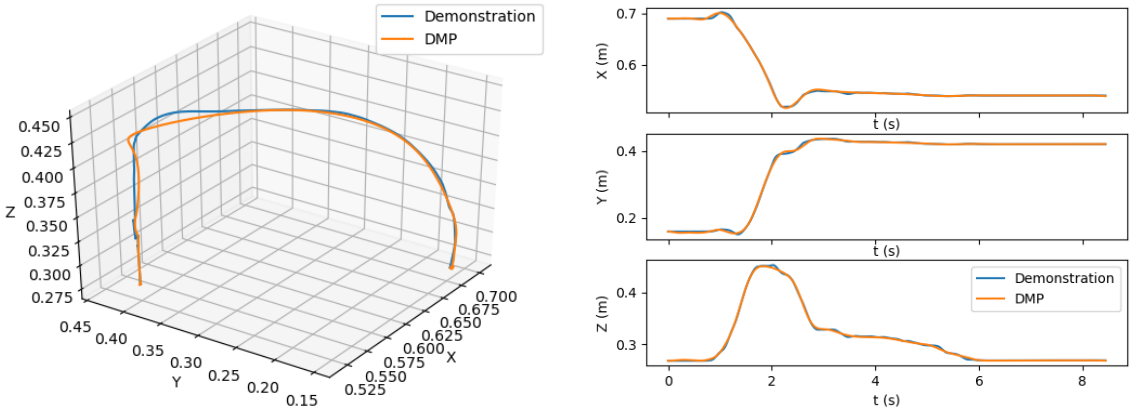
$$\boldsymbol{\Psi W} = \boldsymbol{F} \tag{11}$$

Based on a trial and error approach the following was determined to be used: $\alpha_z = 48$, center of basis function placed with increasing density as χ approaches 0 to accommodate the choice of canonical system. The basis function parameter σ was chosen based on the handed template [2],

$$\sigma_i = c_{i+1} - c_i \tag{12}$$

Which indicates that the width of the basis functions decreases as the density increases.

By fitting the DMPs to the demonstration in Figure 1, the following is achieved



(a) Dynamic Movement Primitive fitted to the demonstration shown in cartesian space.

(b) DMP fitted to the demonstration shown in X, Y and Z axes.

Figure 3: Results for fitting the DMP to the demonstration with $\tau = \frac{1}{500}N_{demo}$, where N_{demo} is amount of samples in demonstration, and with 30 basis functions.

2.1 Including Rotation in the DMP

Bjarke Larsen

In addition to controlling the end effector position as presented in section 2 it is possible to also control the orientation of the end effector in a similar way. Here we will focus on the use of quaternions for describing the orientation of the end effector since this gives a robust representation with few variables. As presented in [3], analogous to Equation 1 and 2 a system can be set up. In the following $\bar{\mathbf{q}}$ denotes the quaternion conjugate and * denotes the quaternion product.

$$\tau\dot{\boldsymbol{\eta}} = \alpha_z(\beta_z \cdot 2 \cdot log(\mathbf{g}_0 * \bar{\mathbf{q}}) - \boldsymbol{\eta}) + \mathbf{f}(x) \tag{13}$$

$$\tau\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\eta} * \mathbf{q} \tag{14}$$

Where Equation 14 has to be integrated in the following way [3].

$$\mathbf{q}(t + \delta t) = exp\left(\frac{\delta t}{2}\frac{\boldsymbol{\eta}(t)}{\tau}\right) * \mathbf{q}(t) \tag{15}$$

The main reason the system in Equations 13 and 14 differs from what seen previously is that while the variables for position are independent they are not independent for rotations. This means that learning the weights for the basis functions as seen in Equation 8 is written on the following form.

$$\tau^2\dot{\boldsymbol{\omega}}_{demo} - \alpha_z(\beta_z \cdot 2 \cdot log(\mathbf{g}_0 * \bar{\mathbf{q}}_{demo}) - \tau \cdot \boldsymbol{\omega}_{demo}) = \mathbf{f}_{target} \tag{16}$$

Here $2log(\mathbf{g}_0 * \bar{\mathbf{q}}_{demo})$ is the angular velocity in unit time between the current demonstrated $\mathbf{q}_i$ and the goal orientation. $\boldsymbol{\omega}_{demo}$ is the angular velocity at the current time calculated as $2log(\mathbf{q}_{i+1} * \mathbf{q}_i)$ and $\dot{\boldsymbol{\omega}}_{demo}$ is the angular acceleration at the current time. Since the angular velocity is given as a quaternion with zero scalar part it is only necessary to learn three weights so the minimization problem is entirely analogous to the problem solved for position in Equation 11. Since the problem of finding the correct weights is the same, $\mathbf{f}$ is the same as in Equation 7 with the only exception being the scaling term $(g_0 - x)$ being a 3x3 matrix with the quaternion velocity.

The result from implementing the DMP for orientation based on modifying the handed template [2] to accommodate quaternions, with 100 basis functions, is seen in Figure 4.

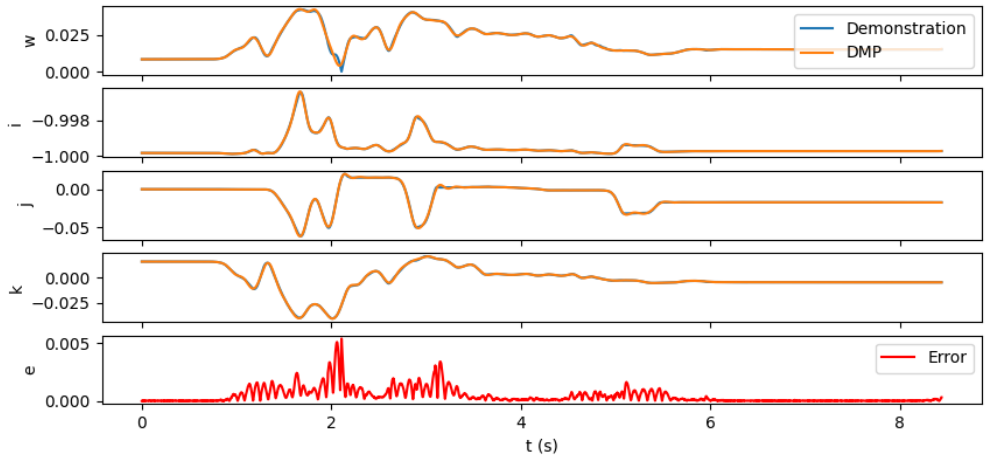


Figure 4: The result from fitting the DMP to the recorded orientation.

The four top plots in Figure 4 are the individual parts of the quaternion, $w + [xi\ yj\ zk]$ The error e, plotted at the bottom in Figure 4 is calculated as the distance on the unit sphere between the recorded data and the fitted data.

3 Potential Fields for Obstacle Avoidance

Emil Vincent Ancker

In this section simple spherical obstacles will be considered. The problem under consideration are obstacles being located on the trajectory of the end-effector, which is visualized in Figure 5.

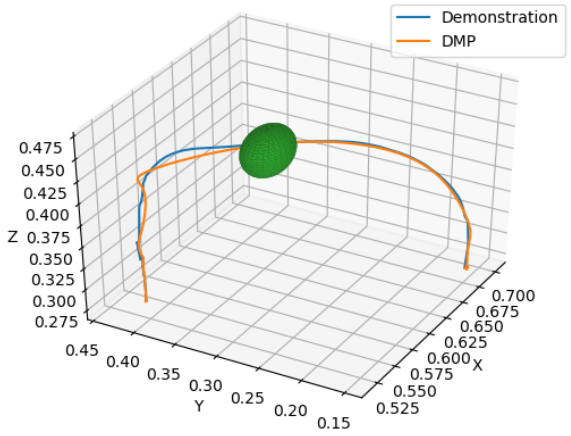


Figure 5: Trajectory demonstration and fitted DMP with introduction of a spherical obstacle.

To avoid collision with the obstacle, the obstacle will need to be incorporated into the trajectory generation.

As stated previously obstacle avoidance can be incorporated into the DMP framework by adding a new term in the transformation system presented in Equation 17. A potential field, C_t, which repels the manipulator away from obstacles is added to the transformation system.

$$\begin{aligned} \tau \dot{\boldsymbol{z}} &= \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{x}) - \boldsymbol{z}) + f + C_t, \\ \tau \dot{\boldsymbol{x}} &= \boldsymbol{z}, \end{aligned} \tag{17}$$

There are many ways to define the potential field, some of these will be implemented and evaluated in this section.

3.1 Potential Field as a Function of Steering Angle

Emil Vincent Ancker

In the work [4] a simple function for human obstacle avoidance is presented:

$$\dot{\theta} = \gamma_0 \theta e^{-\beta_0 |\theta|} \tag{18}$$

Where γ_0 and β_0 are constants and θ is the angle between the velocity of the end effector and the vector extended from the end effector to the obstacle.

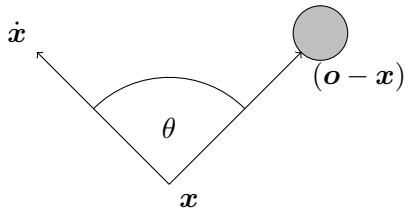


Figure 6: Illustration of steering angle θ, where $\boldsymbol{x}$ is the end-effector position, $\dot{\boldsymbol{x}}$ is the end-effector velocity and the vector $(o - x)$ is the vector from the end-effector to the obstacle.

In [5] a potential field based on Equation 18 is presented:

$$C_t = \gamma_0 R\dot{\boldsymbol{x}}\theta e^{-\beta_0|\theta|} \tag{19}$$

The steering angle in Figure 6 can simply be calculated by finding the angle between the vectors $\boldsymbol{(o-x)}$ and $\dot{\boldsymbol{x}}$ on the plane where the vectors lie.

$$\theta = \cos^{-1}\left(\frac{\boldsymbol{(o-x)}^T\dot{\boldsymbol{x}}}{|\boldsymbol{(o-x)}||\dot{\boldsymbol{x}}|}\right) \tag{20}$$

The rotation R is simply a rotation of $\frac{\pi}{2}$ around the normal of the plane where $\boldsymbol{(o-x)}$ and $\dot{\boldsymbol{x}}$ lies:

$$\boldsymbol{n} = \boldsymbol{(o-x)} \times \dot{\boldsymbol{x}} \tag{21}$$

The rotation matrix R can be found from transforming an angle axis rotation of angle $\frac{\pi}{2}$ around the vector $\boldsymbol{n}$:

$$R = [\boldsymbol{n}]_\times + \boldsymbol{n}\boldsymbol{n}^T \tag{22}$$

Where $[\boldsymbol{n}]_\times$ is the skew symmetric matrix of $\boldsymbol{n}$.

3.1.1 Evaluation of Steering Angle Based DMP

`Emil Vincent Ancker`

When adjusting the tuning parameters of the potential field (γ, β), there are multiple external factors which also have an effect. The end-effector velocity $\dot{\boldsymbol{x}}$ has a direct effect on the choice of γ, since following a trajectory with higher velocity would lead to a repulsive force of higher magnitude shown in Equation 18.

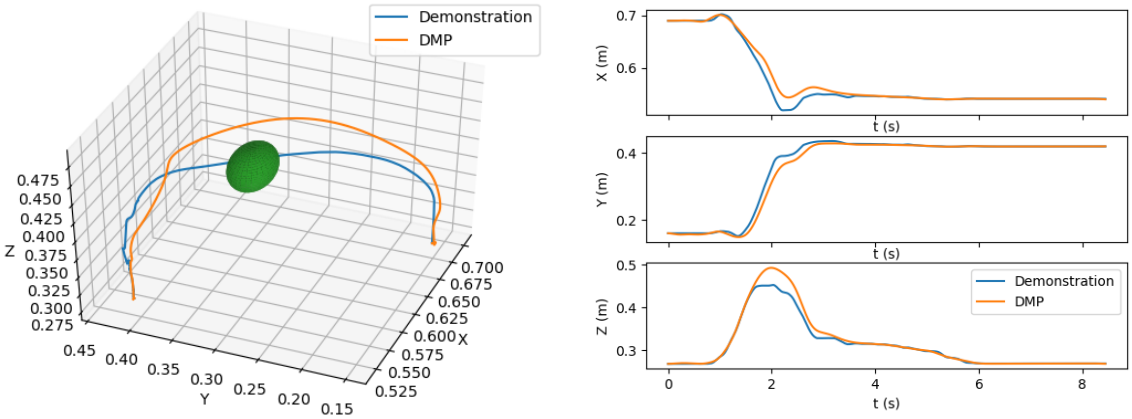
$$\uparrow \dot{\boldsymbol{x}} \quad \Rightarrow \quad \downarrow \gamma_0 \tag{23}$$

Furthermore, the shape of the obstacle has an influence on the tuning of the parameters. The parameter β determine the sensitivity of the steering angle:

$$\uparrow \beta_0 \quad \Rightarrow \quad \text{lower sensitivity span of steering angle} \tag{24}$$

$$\downarrow \beta_0 \quad \Rightarrow \quad \text{higher sensitivity span of steering angle} \tag{25}$$

In other words, if β is chosen high, the DMP only avoids the obstacle if it is almost directly in front of it. But if β is chosen too low it might incorporate obstacles in the motion that are too far to the side.

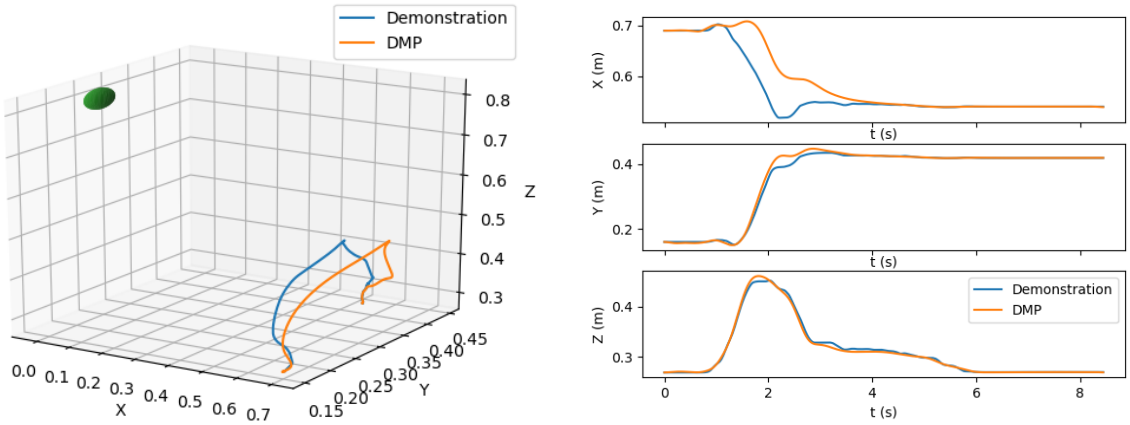


(a) Trajectories visualized in cartesian space.

(b) Trajectory shown in X, Y and Z axes.

Figure 7: Results for using a potential fields which only takes steering angles into account with the following tuning variable values: $\gamma_0 = 10^6$, $\beta_0 = \frac{20}{\pi}$. Obstacle is a sphere with $r = 0.025[m]$ located at $xyz = [0.0, 0.25, 0.80]$.

An example of parameters that is well adjust to the demonstration is shown in Figure 7. But there are numerous problems with this potential field, which can be seen from Equation 18. If the end-effector is heading directly towards the obstacle, $\theta = 0$, then it will not get repelled by the potential field since the magnitude is zero. Furthermore, obstacles far away have an influence on the motion, which is shown in Figure 8.



(a) Trajectories visualized in cartesian space.

(b) Trajectory shown in X, Y and Z axes.

Figure 8: Problem of obstacles far away visualized with the following tuning variable values: $\gamma_0 = 10^6$, $\beta_0 = \frac{20}{\pi}$. Obstacle is a sphere with $r = 0.025[m]$ located at $xyz = [0.0, 0.25, 0.80]$.

3.2 Potential Fields with Coupled Terms

`Mikkel Larsen`

In order to counteract the problems stated above where when the angle $\theta = 0$ the repelling force yielded $C_t = 0$ and the potential field always has an impact on the motion no matter the distance, the paper [5] presented a new potential field which is shown in Equation 26.

$$C_t = \gamma_1 R\dot{\boldsymbol{x}}\phi_1 + \gamma_2 R\dot{\boldsymbol{x}}\phi_2 + \gamma_3 R\dot{\boldsymbol{x}}\phi_3 \tag{26}$$

Furthermore the new potential field does also consider nearest point on the obstacle seen from the end-effector instead of just using the center of the obstacle. This is particularly important when dealing with non-uniform obstacles, because only using the center of the obstacle could lead to collision. The three coupling terms ϕ_1, ϕ_2, ϕ_3 are defined as

$$\phi_1 = \theta e^{-\beta_0|\theta|} \cdot e^{-kd} \tag{27}$$

$$\phi_2 = \theta e^{-\beta_0|\theta_p|} \cdot e^{-kd_p} \tag{28}$$

$$\phi_3 = e^{-kd} \tag{29}$$

Where k is a constant, d is the distance to the center of the object from the end-effector, θ_p is the angle from the end-effector to the nearest point on the object and d_p is the distance from the end-effector to the nearest point on the object. From the coupling terms it can be seen the term e^{-kd} scales the potential field with the distance, where the longer away from the obstacle the less effect from the repulsive force. This results in ϕ_1 and ϕ_2 are

repulsive forces away from the obstacle center and nearest point on the object respectively taking the angle θ into account, and ϕ_3 is a repulsive force which is used when the angle $\theta = 0$.

3.2.1 Evaluation of Potential Fields with Coupled Terms DMP

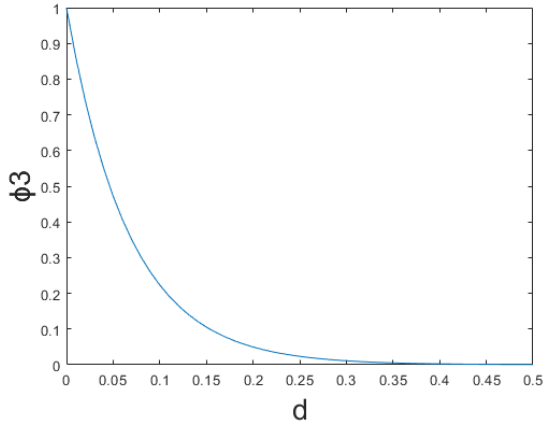
`Mikkel Larsen`

The following parameters ($\gamma_1, \gamma_2, \gamma_3$, k) of the potential field with coupled terms needs to be adjusted, where $\dot{\boldsymbol{x}}$ still has the same effect on γ_i as described in subsubsection 3.1.1. The parameter k scales the distance to the object meaning:

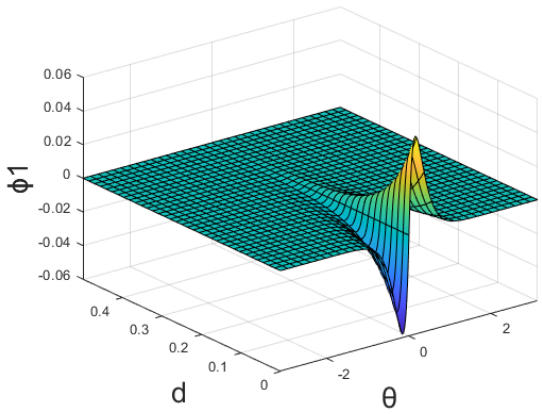
$$\uparrow k \quad \Rightarrow \quad \text{Lesser impact of the distance to the object} \tag{30}$$

$$\downarrow k \quad \Rightarrow \quad \text{Higher impact of the distance to the object} \tag{31}$$

This means if k is chosen to be high, the DMP can get closer to the object before the repulsive field has an impact and the opposite effect if k is chosen to be low. With $\beta_0 = \frac{20}{\pi}$ and $k = 15$ the repulsive field from ϕ_1 and ϕ_3 is activated around a distance of $d = 0.3$ which was deemed acceptable. This is illustrated in Figure 9.



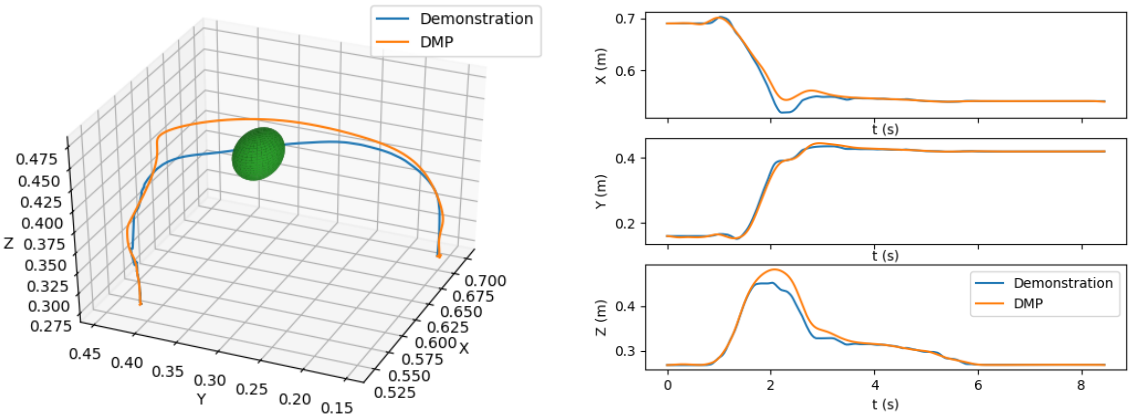
(a) Coupling term ϕ_3 versus distance to object center d.



(b) Coupling term ϕ_1 versus distance to object center d and angle to object center θ

Figure 9: Plot of coupling term ϕ_3 and ϕ_1 with $k = 15$ and $\beta_0 = \frac{20}{\pi}$.

Next the same performance evaluation is done for DMP with potential fields with coupled terms as done for DMP with steering angle to compare them to each other. However, because the test is done with a sphere as an object, the nearest point on the obstacle does not have an impact and therefore $\gamma_2 = 0$. Figure 10 shows the performance for an sphere with radius $r = 0.025[m]$ located on the trajectory at $xyz = [0.575, 0.30, 0.45]$.



(a) Trajectories visualized in cartesian space.

(b) Trajectory shown in X, Y and Z axes.

Figure 10: Results for using a potential field with coupling terms with the following tuning variable values: $\gamma_1 = 10^6$, $\gamma_2 = 0$, $\gamma_3 = 1.25 \cdot 10^4$, $\beta_0 = \frac{20}{\pi}$, $k = 15$. Obstacle is a sphere with $r = 0.025[m]$ located at $xyz = [0.575, 0.30, 0.45]$.

Looking at the trajectory using coupling terms at Figure 10a compared to the trajectory using only steering angle at Figure 7a it is seen the obstacle avoidance is nearly the same. However, after the obstacle has been avoided, the trajectory using coupling terms is corrected more onto the demonstrated path compared to the DMP trajectory only using steering angle. This is the result of taking the distance to the object into consideration. To further illustrate that the DMP trajectory using coupling terms follows the demonstrated path more than DMP trajectory only using steering angle the euclidean distance between the demonstrated path and the DMP generate path is calculated for both the coupling term DMP and the steering angle DMP. This is illustrated in Figure 11 where the same sphere parameters and DMP parameters are used as in Figure 10 and Figure 7 for coupling term DMP and steering angle DMP respectively.

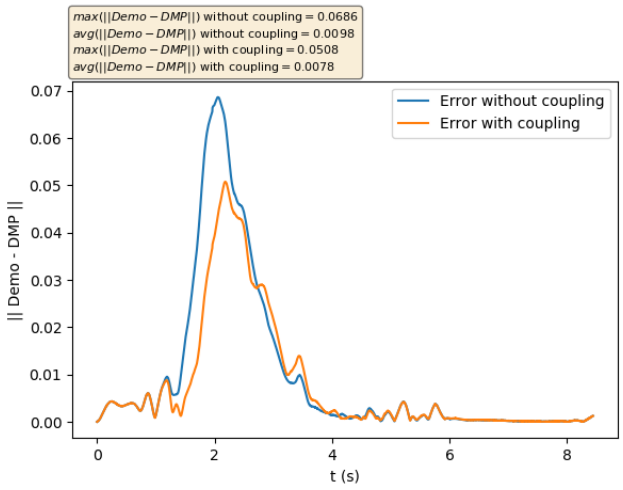
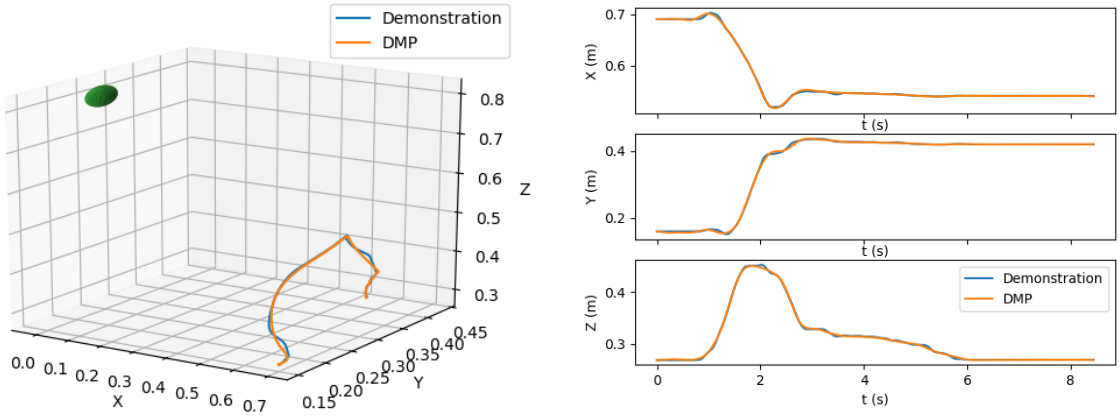


Figure 11: Euclidean distance between the demonstrated path and the DMP generated paths for DMP with coupling terms and DMP with steering angle. Path for DMP with coupling terms is the same as in Figure 10 and path for DMP with steering is the same as in Figure 7.

Figure 11 confirms that the DMP with coupling terms follows the demonstrated path more accurate than the DMP with steering angle. Furthermore, it is seen that the obstacle is avoided around the 2 seconds mark because of the high spike in difference between demonstrated path and DMP path. The DMP with coupling terms has 25.9% smaller maximum euclidean distance and 25.6% smaller average euclidean distance compared to DMP with steering angle.

The performance when an obstacle is away from the trajectory is illustrated in Figure 12. Compared to DMP only using steering angle shown in Figure 8 the DMP trajectory is untouched. Again this is the result of taking the distance to the object into consideration.



(a) Trajectories visualized in cartesian space.

(b) Trajectory shown in X, Y and Z axes.

Figure 12: Results for using a potential field with coupling terms with the following tuning variable values: $\gamma_1 = 10^6$, $\gamma_2 = 0$, $\gamma_3 = 1.25 \cdot 10^4$, $\beta_0 = \frac{20}{\pi}$, $k = 15$. Obstacle is a sphere with $r = 0.025[m]$ located at $xyz = [0.0, 0.25, 0.80]$.

3.3 Online Obstacle Avoidance

`Mikkel Larsen`

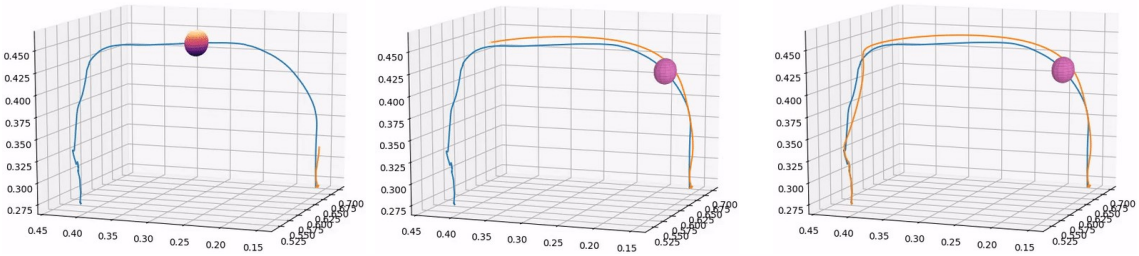
In order to make the robot able to avoid moving obstacles online obstacle avoidance is implemented. The main difference between DMP with a static object and DMP with a moving object is that the obstacles position needs to be updated at each step of calculating the DMP.

3.3.1 Evaluation of Online Obstacle Avoidance

`Mikkel Larsen`

From subsubsection 3.2.1 it was described that DMP with coupling terms were superior compared to DMP with steering angle. Thus, a test is conducted with DMP with coupling terms using the same tuning variable values in subsubsection 3.2.1. Furthermore, the sphere's radius is set to $r = 0.015[m]$ and the starting point and end point of the sphere is $start_{xyz} = [0.53, 0.39, 0.45]$ and $end_{xyz} = [0.65, 0.17, 0.41]$ respectively and the sphere follows the demostrated path. The online path generated is illustrated in Figure 13 and visualized on Github[1].

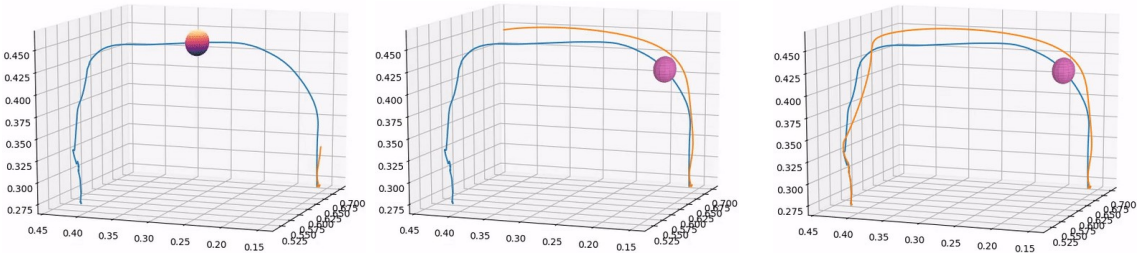
[1] `https://github.com/Masle16/obstacle_avoidance_with_dmps/tree/dmp_online_obs_avoid/DMP`



(a) Frame 40 out of 281 (b) Frame 70 out of 281 (c) Frame 281 out of 281

Figure 13: Three steps of online obstacle avoidance using same tuning variable values as in subsubsection 3.2.1. Sphere's radius is $r = 0.015[m]$, sphere's start point and end point is $start_{xyz} = [0.53, 0.39, 0.45]$ and $end_{xyz} = [0.65, 0.17, 0.41]$. Sphere follows the demonstrated path and stops moving when purple.

Figure 13 shows that the obstacle is not avoided thus the decouple terms are adjusted. The tuning variable values are modified to $\gamma_1 = 3 \cdot 10^6$ and $\gamma_3 = 2 \cdot 10^4$ in order to avoid the obstacle. The following tuning variable values can be seen in Figure 14 and visualized on Github[2]



(a) Frame 40 out of 281 (b) Frame 70 out of 281 (c) Frame 281 out of 281

Figure 14: Three steps of online obstacle avoidance using tuning variable values: $\gamma_1 = 3 \cdot 10^6$, $\gamma_2 = 0$, $\gamma_3 = 2 \cdot 10^4$, $\beta_0 = \frac{20}{\pi}$, $k = 15$ in order to avoid the obstacle. Sphere's radius is $r = 0.015[m]$, sphere's start point and end point is $start_{xyz} = [0.53, 0.39, 0.45]$ and $end_{xyz} = [0.65, 0.17, 0.41]$. Sphere follows the demonstrated path and stops moving when purple.

3.4 Attraction Field Towards Demonstration

`Emil Vincent Ancker`

To pull the end-effector back towards the demonstrated path after an obstacle has been encountered an attractional potential field is added to the demonstration. The potential field is defined to be a relationship between the positional error $\boldsymbol{x}_{demo} - \boldsymbol{x}$ and the euclidean distance to the obstacle $||(\boldsymbol{o} - \boldsymbol{x})||$.

[2] `https://github.com/Masle16/obstacle_avoidance_with_dmps/tree/dmp_online_obs_avoid/DMP`

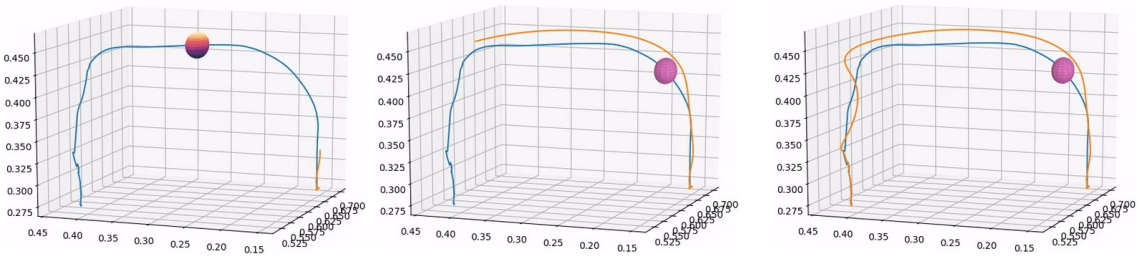
$$U(\boldsymbol{x}) = \begin{cases} \gamma_T(\boldsymbol{x}_{demo} - \boldsymbol{x})e^{||(\boldsymbol{o}-\boldsymbol{x})||}, & ||(\boldsymbol{o}-\boldsymbol{x})|| > d_T \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

Where γ_T is a constant defined the strength of the potential field and d_T is a radius of non-influence meaning that within a certain distance of the obstacle the attraction towards the demonstrated trajectory should be ignored.

3.4.1 Evaluation of Attraction Field

`Mikkel Larsen`

The same path for both the DMP and the moving obstacle is used when evaluating the DMP with attraction field as in Figure 14. Additionally the same tuning variables are kept as in Figure 14. The threshold for when to activate the attractor field is set to $d_T = 0.05$ and the potential attractor field strength is set to $\gamma_T = 2.5 \cdot 10^3$. This yields the following result shown in Figure 15



(a) Frame 40 out of 281 (b) Frame 70 out of 281 (c) Frame 281 out of 281

Figure 15: Three steps of online obstacle avoidance with attractor field using the same tuning variable values as in Figure 14. The attractor field parameters were set to $d_T = 0.05$ and $\gamma_T = 2.5 \cdot 10^3$. Sphere's radius is $r = 0.015[m]$, sphere's start point and end point is $start_{xyz} = [0.53, 0.39, 0.45]$ and $end_{xyz} = [0.65, 0.17, 0.41]$. Sphere follows the demonstrated path and stops moving when purple.

To quantify the performance of the attractor DMP method versus without the attractor field the euclidean distance between the demonstrated position and DMP position is found. This is illustrated in Figure 16 for the tuned online obstacle avoidance DMP versus the attraction field DMP.

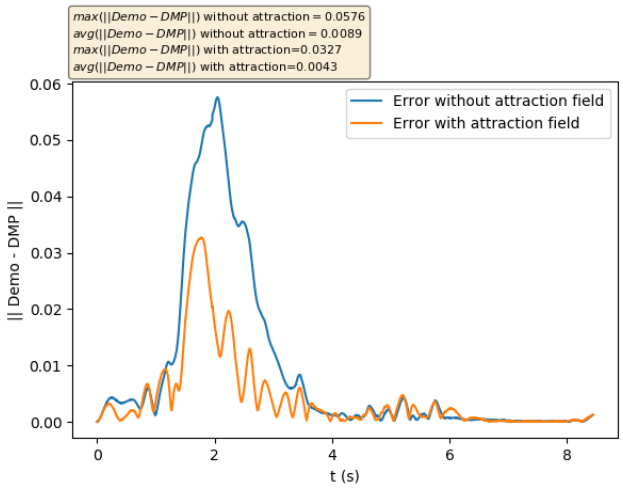


Figure 16: Euclidean distance between the demonstrated path and the DMP generated paths for DMP online obstacle avoidance and DMP with Attraction Field.

From Figure 16 it is seen that using an attractor field while doing online obstacle avoidance makes the DMP generated path follows the demostrated path more accurate compared to not using an attractor field. However, it can also be seen that using an attractor field makes oscillations in the euclidean distance when trying to avoid the obstacle around the 2 seconds mark. The DMP with attractor term has 43.2% smaller maximum euclidean distance and 51.7% smaller average euclidean distance compared to DMP with coupling terms

4 Link Collision Avoidance

Emil Vincent Ancker

The obstacle avoidance until now have only considered end-effector avoidance and since the robot is not just a point in Cartesian space, as considered until now, it is worth looking into link collision avoidance. In this section the robot under study is a redundant manipulator since the work is primarily based on [6], [7] and [8]. The chosen robot is a 7-DOF Franka Emika Panda which will be simulated in Matlab. An example of the robot and a spherical obstacle is shown in Figure 17.

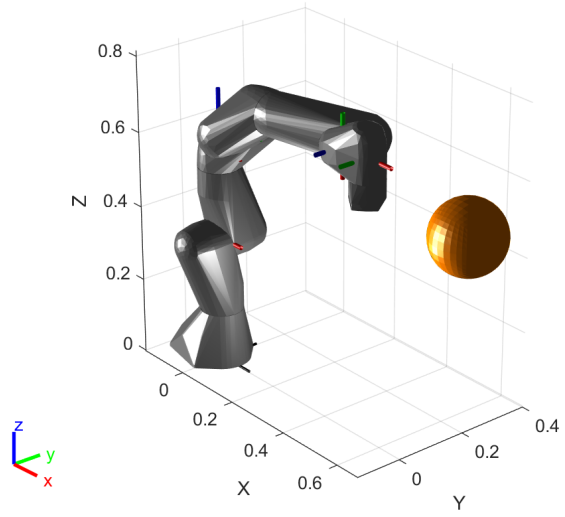


Figure 17: Visualization of the robot and a spherical obstacle.

When performing link collision avoidance an additional repellent is needed to avoid the links of the robot colliding with an obstacle. But instead of incorporating this into the transformation system in Equation 17, it is done through inverse kinematics [7], making the end-effector follow the trajectory generated in previous sections while avoiding link collisions.

4.1 Null Space Movement

Emil Vincent Ancker

The end-effector velocity is related to the joint velocity through the Jacobian:

$$\dot{\boldsymbol{x}}_e = \boldsymbol{J}_e \dot{\boldsymbol{q}} \tag{33}$$

Where $\dot{\boldsymbol{x}}_e$ is the end-effector velocity and $\boldsymbol{J}_e$ is the Jacobian of the end-effector. Since the robot is a redundant manipulator the inverse of the Jacobian can not be found, instead

the pseudoinverse is used to get a least-squares solution:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e \tag{34}$$

Since the robot is a redundant manipulator this also means that there are more solutions to Equation 34, this can be illustrated by introducing a null space movement [6]:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e + \left(\boldsymbol{I} - \boldsymbol{J}_e^+ \boldsymbol{J}_e\right) \boldsymbol{\mathcal{E}} \tag{35}$$

The null space movement term in Equation 35,

$$\left(\boldsymbol{I} - \boldsymbol{J}_e^+ \boldsymbol{J}_e\right) \boldsymbol{\mathcal{E}} \tag{36}$$

Will map the arbitrary velocity, $\boldsymbol{\mathcal{E}}$, into the null space of $\boldsymbol{J}$ [8]. This means that we can now find a joint velocity corresponding to a Cartesian velocity and add an arbitrary joint movement that does not affect the Cartesian velocity.

4.2 Utilizing Null Space Movement for Obstacle Avoidance

`Emil Vincent Ancker`

The foundation of the utilization of null space movement to avoid obstacles builds on Equation 35, where the null space movement is adapted to avoid an obstacle [7].

To incorporate the obstacle avoidance in the null space movement, the point on the robot $\boldsymbol{x}_0$ closest to the obstacle is considered,

$$\dot{\boldsymbol{x}}_0 = \boldsymbol{J}_0 \dot{\boldsymbol{q}} \tag{37}$$

Where $\boldsymbol{J}_0$ is the Jacobian from the base up to $\boldsymbol{x}_0$. By imposing a constraint on the velocity of the robot closest to the obstacle, $\dot{\boldsymbol{x}}_0$, using a potential field it is possible to make this point, $\boldsymbol{x}_0$, move away from the obstacle. A potential field $U(\boldsymbol{x})$ is now considered [7]:

$$U(\boldsymbol{x}) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{p(\boldsymbol{x})} - \frac{1}{p_0}\right)^2, & \text{if } p(\boldsymbol{x}) < p_0 \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

The potential field in Equation 38 repels the point $\boldsymbol{x}$ away from the obstacle if it is within a threshold distance p_0. The term $p(\boldsymbol{x})$ is the distance from $\boldsymbol{x}$ to the obstacle. The repulsive

effect is implemented by,

$$\dot{\boldsymbol{x}}_0 = -\gamma_L \nabla U(\boldsymbol{x}_0), \tag{39}$$

$$\nabla_{\boldsymbol{x}} U(\boldsymbol{x}) = \eta \left(\frac{1}{p(\boldsymbol{x})} - \frac{1}{p_0} \right) \left(\frac{-1}{p(\boldsymbol{x})^2} \right) \nabla_{\boldsymbol{x}} p(\boldsymbol{x}) \tag{40}$$

Where γ_L is a constant that has to be tuned. The derivative of the potential field is only as stated above when $p(\boldsymbol{x}) < p_0$.

By substituting Equation 35 into Equation 37, the following is obtained:

$$\dot{\boldsymbol{x}}_0 = \boldsymbol{J}_0 \left(\boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e + \left(\boldsymbol{I} - \boldsymbol{J}_e^+ \boldsymbol{J}_e \right) \boldsymbol{\mathcal{E}} \right) \tag{41}$$

Since all entities in the above is known except the arbitrary velocity $\boldsymbol{\mathcal{E}}$, it can be isolated and the null-space velocity is obtained,

$$\boldsymbol{\mathcal{E}} = \left(\boldsymbol{J}_0 (I - \boldsymbol{J}_e^+ \boldsymbol{J}_e) \right)^+ (\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0 \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e) \tag{42}$$

The expression for the null-space velocity, $\boldsymbol{\mathcal{E}}$, is now substituted into Equation 35,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e + \left(\boldsymbol{I} - \boldsymbol{J}_e^+ \boldsymbol{J}_e \right) \left(\boldsymbol{J}_0 (I - \boldsymbol{J}_e^+ \boldsymbol{J}_e) \right)^+ (\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0 \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e) \tag{43}$$

According to [6] Equation 43 can be simplified to the following,

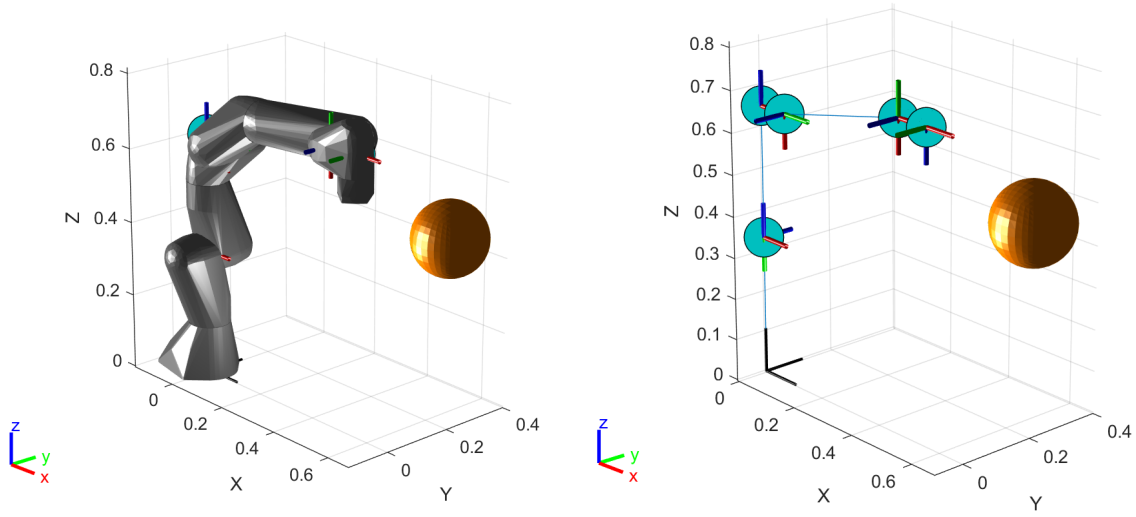
$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e + \left(\boldsymbol{J}_0 (I - \boldsymbol{J}_e^+ \boldsymbol{J}_e) \right)^+ (\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0 \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e) \tag{44}$$

Everything is now known and therefore the link collision avoidance can now be implemented.

4.3 Implementation of Pseudoinverse based Link Collision Avoidance

`Emil Vincent Ancker`

In Equation 44 it is necessary to find the closest point $\boldsymbol{x}_0$ to determine both $\boldsymbol{J}_0$ and $\dot{\boldsymbol{x}}_0$. It was chosen to discretize the robot into 7 points located at each joint, which can be seen in Figure 18.



(a) Robot with body meshes shown and indication of points.

(b) Robot without body meshes shown and indication of points.

Figure 18: Robot shown with and without body meshes and with indication of points to show where the points that are being observed are located on the robot.

The point $\boldsymbol{x}_0$ is set to the point on the robot which is closest to the obstacle. The procedure at each time step for the link collision avoidance is shown in algorithm 1, the trajectory given as input $\boldsymbol{T}$ might change while the robot is evaluating according to the online obstacle avoidance in subsection 3.3.

Algorithm 1: Link collision avoidance

Requires initialization of: $\boldsymbol{T}$ (trajectory), Δt (time step), $\boldsymbol{q}$ (initial configuration).

while *points left in* $\boldsymbol{T}$ **do**

$\dot{\boldsymbol{x}}_e = \frac{\boldsymbol{T}_{i+1} - \boldsymbol{T}_i}{\Delta t}$

$\boldsymbol{x}_0 = \arg\min_{\boldsymbol{x}} ||(\boldsymbol{o} - \boldsymbol{x}_i)||^2$ ▷ find point on robot closest to obstacle

$\dot{\boldsymbol{x}}_0 = -\gamma_L \nabla U(\boldsymbol{x}_0)$

$\boldsymbol{J}_0 = \text{ComputeJacobian}(\boldsymbol{x}_0)$

$\boldsymbol{J}_e = \text{ComputeJacobian}(\boldsymbol{x}_e)$

$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e + \left(\boldsymbol{J}_0 (I - \boldsymbol{J}_e^+ \boldsymbol{J}_e)\right)^+ (\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0 \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e)$

$\boldsymbol{q} = \boldsymbol{q} + \dot{\boldsymbol{q}} \Delta t$ ▷ configuration is being updated

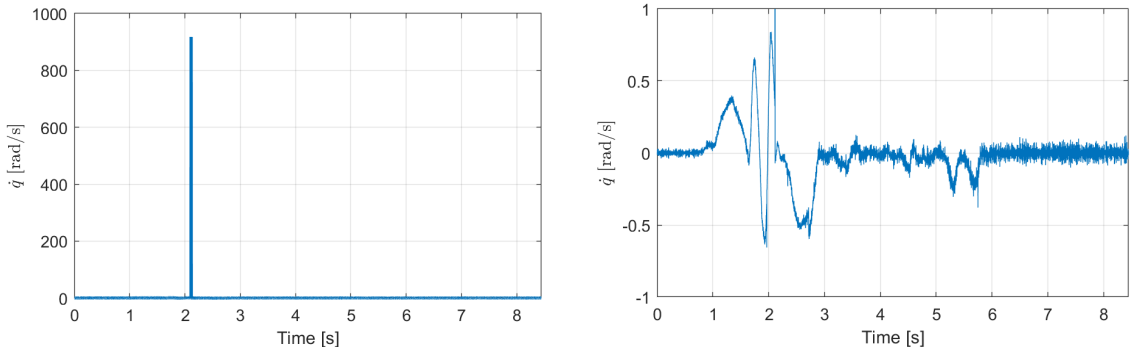
end

By using the computed joint velocities in algorithm 1 which is based on [6] and [7] multiple problems were observed.

The first problem observed was that the term $(J_0(I - J_e^+ J_e))^+$ often becomes singular or close to, which produces very high joint velocities, this problem could simply be avoided by

introducing thresholding on the Moore-Penrose inverse. Another problem was observed, but this was with respect to the term $\boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e$, which produces very high joint velocities when the robot is close to a singularity.

The problems can be illustrated by giving the trajectory generated by the DMP as input, and inspecting the joint velocities for the third joint from the base shown in Figure 19.



(a) Joint velocities for the third joint directly obtained from the inverse kinematics solution.

(b) Joint velocities for the third joint with limits on the y-axis of $[-1,\ 1]$.

Figure 19: Joint velocities obtained from inverse kinematics with null space movement.

In Figure 19a it can be observed that approximately two seconds into the execution of the trajectory, the inverse kinematics leads to joint velocity of infeasibly large magnitude. The resulting trajectory can be seen in Figure 20.

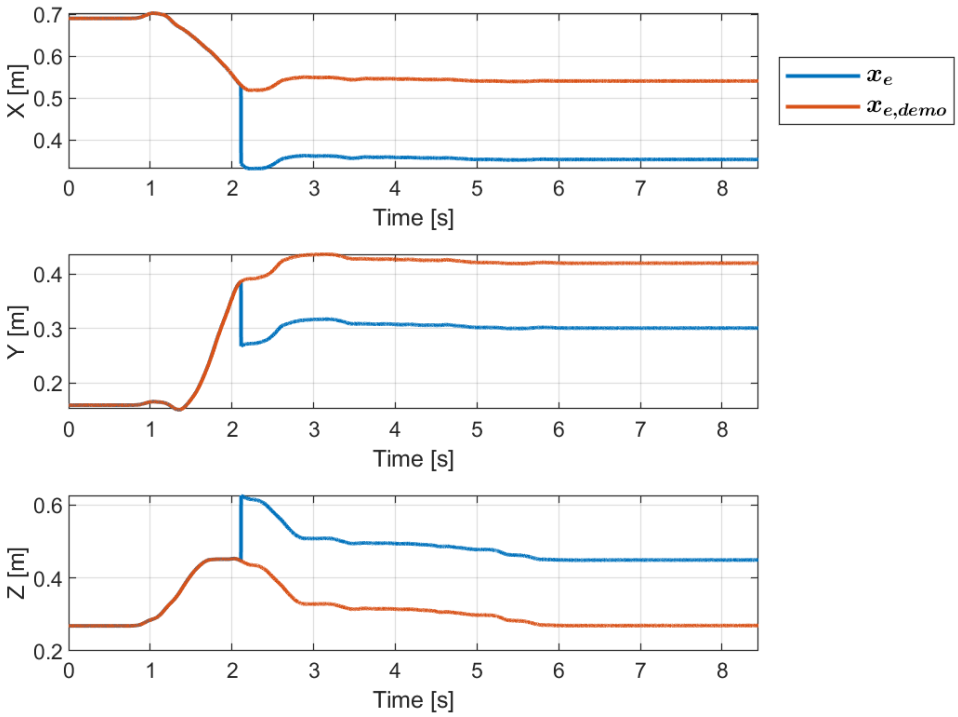


Figure 20: Resulting trajectory from the generated joint velocities from inverse kinematics based on pseudoinverse Jacobian.

The problem of inverse kinematics based on pseudoinverse observed in Figure 19 and Figure 20 is also stated in [8], where damped least squares for inverse kinematics is suggested to handle configurations close to singularities better.

4.4 Damped Least Squares for Inverse Kinematics

`Emil Vincent Ancker`

The inverse kinematics problem is stated as a minimization problem, where the objective is to minimize the error $\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_e$ and punish the solution by the magnitude of $\dot{\boldsymbol{q}}$. The minimization problem is as follows [8],

$$\min\ f(\dot{\boldsymbol{q}}) = ||\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_e||^2 + \lambda^2||\dot{\boldsymbol{q}}||^2 \tag{45}$$

Where $\lambda \in \mathbb{R}$ is a non-zero damping constant [8]. The solution minimizing the problem in Equation 45 is as follows [8],

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T + \lambda^2\boldsymbol{I})^{-1}\dot{\boldsymbol{x}}_e \tag{46}$$

The solution in Equation 46 can also be found by formulating the solution as a system of linear equations,

$$(\boldsymbol{J}_e\boldsymbol{J}_e^T + \lambda^2\boldsymbol{I})\boldsymbol{f} = \dot{\boldsymbol{x}}_e \tag{47}$$

Once the solution to the linear system in Equation 47 have been found the joint velocities can be found,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^T\boldsymbol{f} \tag{48}$$

Finally, the null space movement is added to the found joint velocity in Equation 48 to enable link collision avoidance,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_e^T\boldsymbol{f} + \left(\boldsymbol{J}_0(I - \boldsymbol{J}_e^+\boldsymbol{J}_e)\right)^+(\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0\boldsymbol{J}_e^+\dot{\boldsymbol{x}}_e) \tag{49}$$

By following the same trajectory as done for the pseudoinverse based inverse kinematics it results in the trajectory shown in Figure 21.

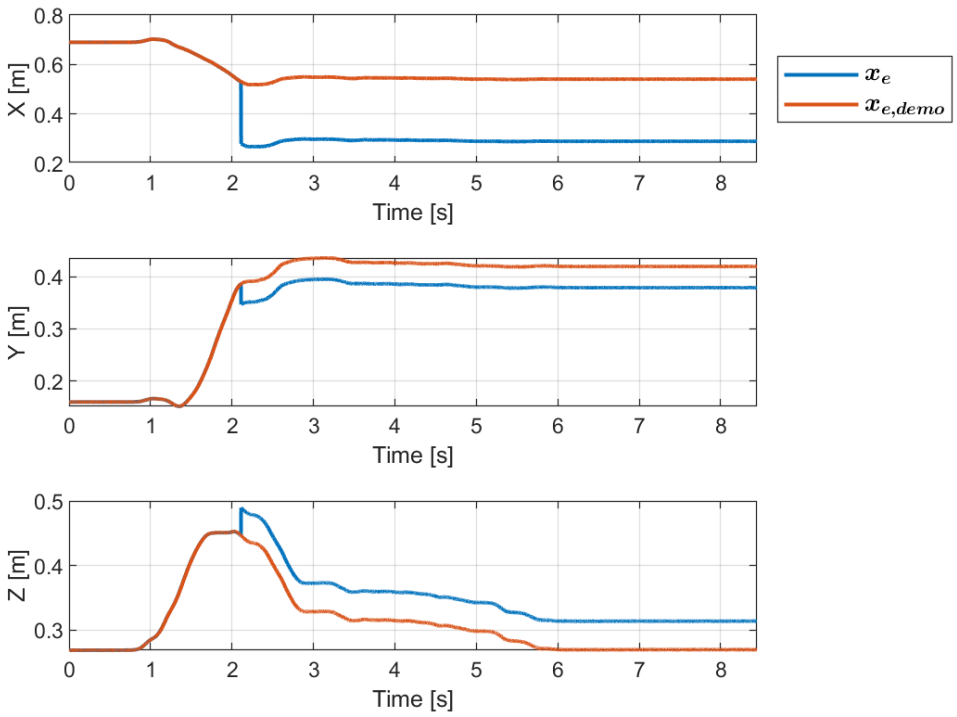


Figure 21: Resulting trajectory from the generated joint velocities from inverse kinematics based on damped least squares. With damping coefficient $\lambda = 0.05$.

It can be seen from the generated trajectory, that the solution found by Equation 49 is less sensitive to singularities, but it does still have an impact on the path.

4.4.1 Attracting the End-effector

`Emil Vincent Ancker`

Instead of using the end-effector velocity generated from the DMPs, $\dot{\boldsymbol{x}}_{DMP}$, directly a trajectory error is defined,

$$\dot{\boldsymbol{x}}_e = \frac{\boldsymbol{x}_{e,demo} - \boldsymbol{x}_e}{dt}, \quad dt = \frac{1}{500} \tag{50}$$

By directly introducing this into the solution, the response in Figure 21 is obtained.

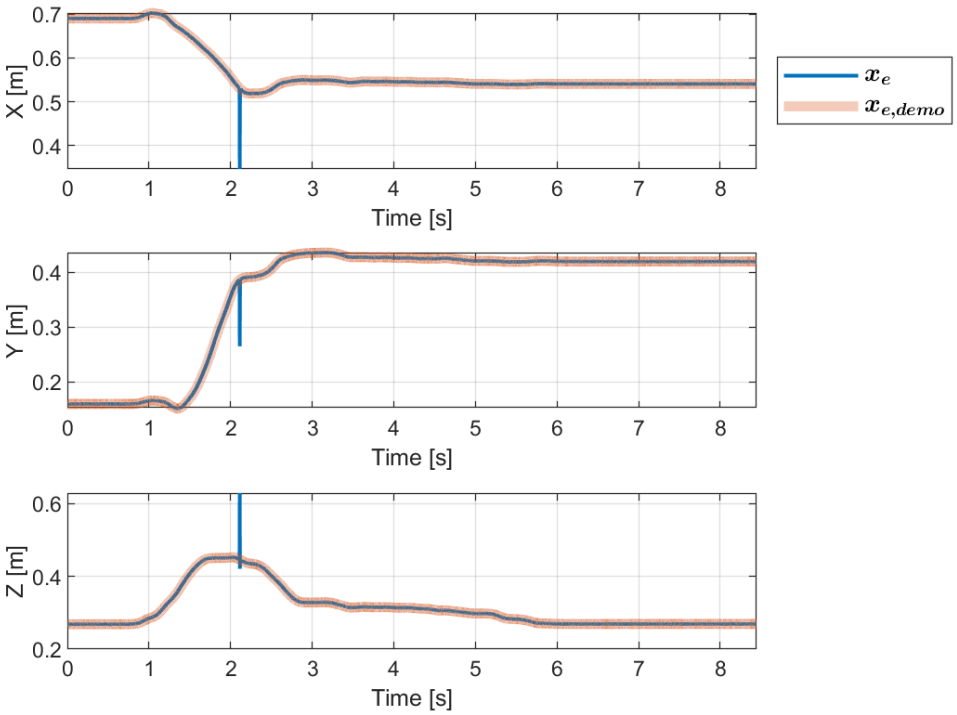


Figure 22: Resulting trajectory from the generated joint velocities from inverse kinematics based on damped least squares while taking current end-effector position into account. With damping coefficient $\lambda = 0.05$.

The resulting joint velocity is still infeasibly high, and might be improved by using `ClampAbsMax`, which is shown in Equation 51.

$$\text{ClampAbsMax}(\dot{\boldsymbol{q}}, d) = \begin{cases} \dot{\boldsymbol{q}}, & ||\dot{\boldsymbol{q}}|| < d \\ d\frac{\dot{\boldsymbol{q}}}{||\dot{\boldsymbol{q}}||}, & \text{otherwise} \end{cases} \tag{51}$$

By utilizing ClampAbsMax the resulting path in Figure 23 is obtained.

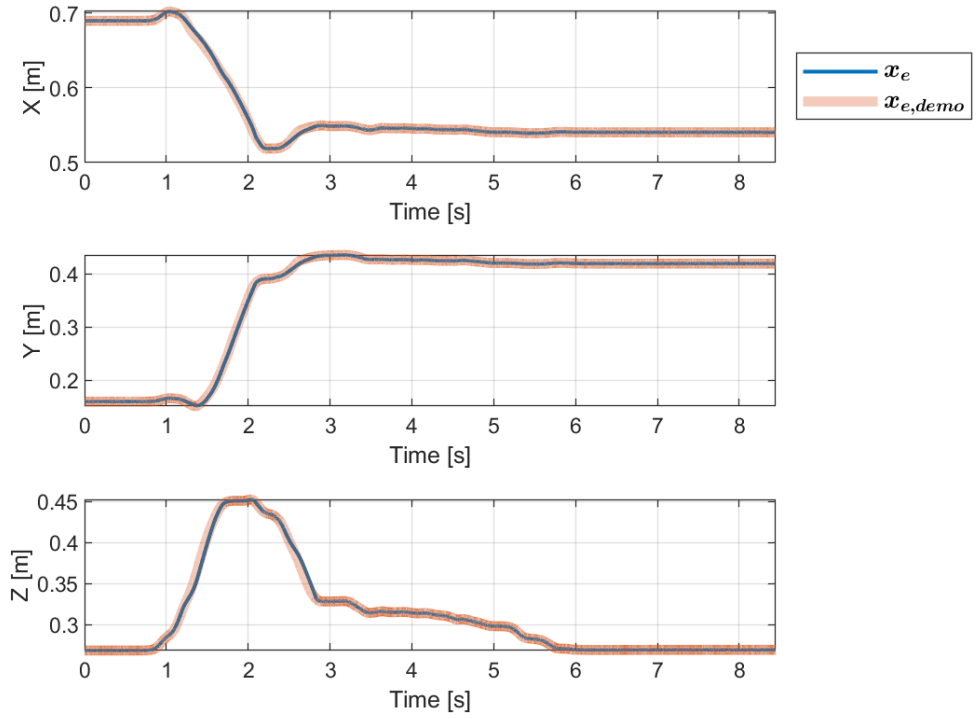


Figure 23: Resulting trajectory from the generated joint velocities from inverse kinematics based on damped least squares while taking current end-effector position into account and utilizing ClampAbsMax with $d = 1[rad/s]$ and damping coefficient $\lambda = 0.05$.

An example of the joint velocities is shown in Figure 24, where the joint velocities for the third joint is shown.

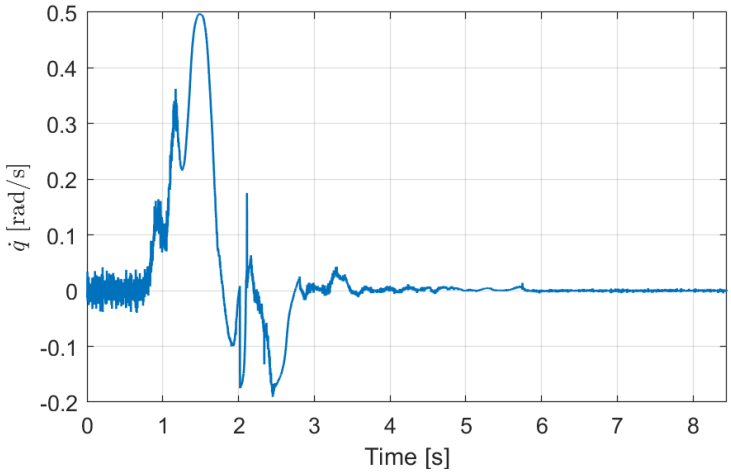


Figure 24: Resulting joint velocity for the third joint from inverse kinematics based on damped least squares while taking current end-effector position into account and utilizing ClampAbsMax with $d = 1[rad/s]$ and damping coefficient $\lambda = 0.05$.

4.4.2 Introducing an Obstacle

By setting the strength of the potential field to $\gamma_L = 1.0$ and introducing an obstacle at location $\boldsymbol{o} = (0.25,\ 0.25,\ 0.45)^T$ the response in Figure 25.

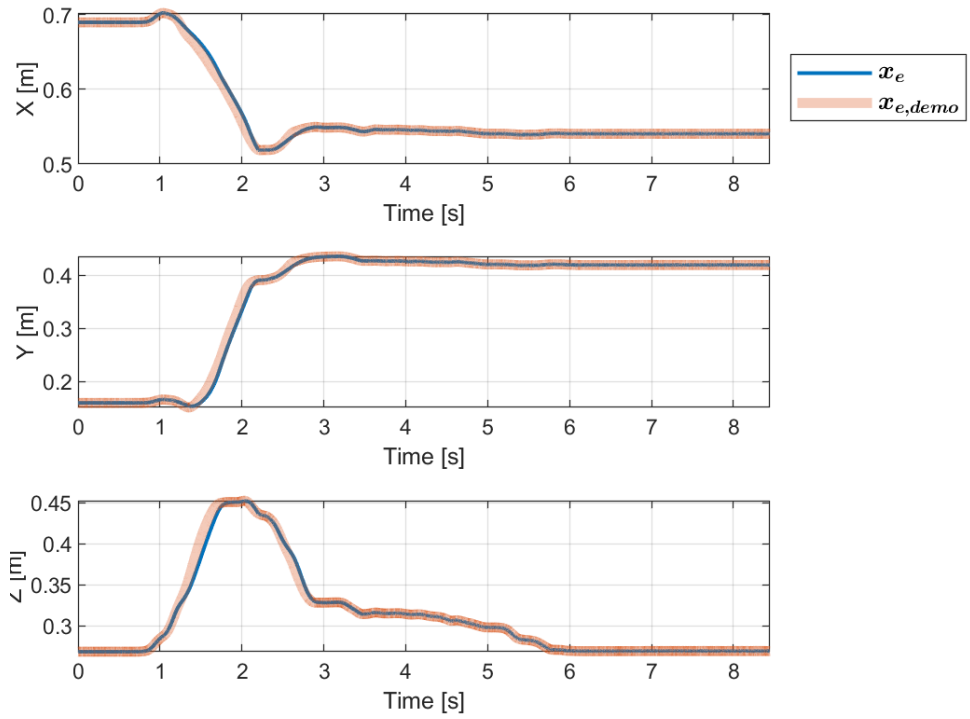


Figure 25: Link collision avoidance performed using inverse kinematics based on damped least squares utilizing ClampAbsMax with $d = 1[rad/s]$, damping coefficient $\lambda = 0.05$ and potential field strength $\gamma_L = 1.0$.

It can be seen in Figure 25 that there is a negligible small deviation from the path. Visualization of a link collision can be seen in Figure 26, while a link collision avoidance can be seen in Figure 27. Animations showing multiple examples of link collision and link collision avoidance are available on the project's Github repository[3].

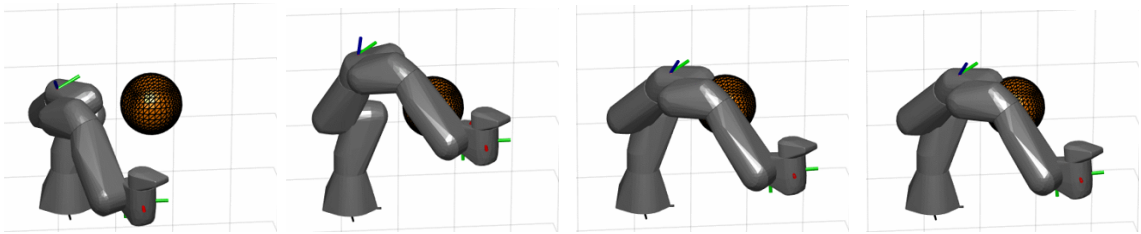


Figure 26: Example of a link collision. Strength of potential field is set to $\gamma_L = 0$.

[3] `https://github.com/Masle16/obstacle_avoidance_with_dmps/tree/master/LinkCollisionAvoidance/GIF`

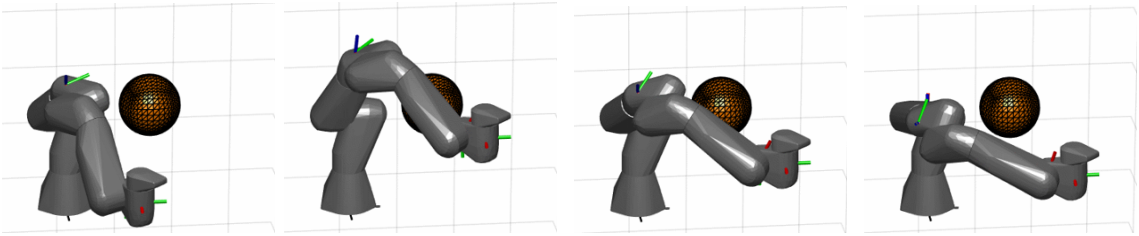


Figure 27: Example of link collision avoidance. Strength of potential field is set to $\gamma_L = 100$.

An example of the resulting joint velocities is once again shown for the third joint in Figure 28, where it clearly can be seen that the joint has a new velocity profile to incorporate the link collision avoidance.

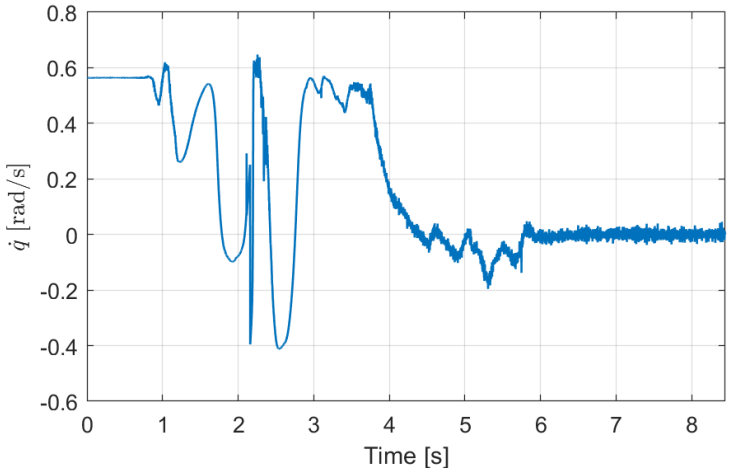


Figure 28: Resulting joint velocity for the third joint when performing link collision avoidance using inverse kinematics based on damped least squares while taking current end-effector position into account and utilizing ClampAbsMax with $d = 1[rad/s]$, damping coefficient $\lambda = 0.05$ and potential field strength $\gamma_L = 1.0$.

The final procedure at each time step when performing link collision avoidance is shown in algorithm 2.

Algorithm 2: Link collision avoidance using IK solved with damped least squares

Requires initialization of: $\boldsymbol{T}$ (trajectory), Δt (time step), $\boldsymbol{q}$ (initial configuration).

while *points left in* $\boldsymbol{T}$ **do**

1 $\quad \dot{\boldsymbol{x}}_e = \frac{\boldsymbol{T}_{i+1} - \boldsymbol{x}_e}{\Delta t}$

2 $\quad \boldsymbol{x}_0 = \arg\min_{\boldsymbol{x}} ||(\boldsymbol{o} - \boldsymbol{x}_i)||^2$ ▷ find point on robot closest to obstacle

3 $\quad \dot{\boldsymbol{x}}_0 = -\gamma_L \nabla U(\boldsymbol{x}_0)$

4 $\quad \boldsymbol{J}_0 = \text{ComputeJacobian}(\boldsymbol{x}_0)$

5 $\quad \boldsymbol{J}_e = \text{ComputeJacobian}(\boldsymbol{x}_e)$

6 $\quad \boldsymbol{f} = \text{LinearSolve}\big((\boldsymbol{J}_e \boldsymbol{J}_e^T + \lambda^2 \boldsymbol{I})\boldsymbol{f} = \dot{\boldsymbol{x}}_e\big)$

7 $\quad \dot{\boldsymbol{q}} = \boldsymbol{J}_e^T \boldsymbol{f} + \big(\boldsymbol{J}_0 (I - \boldsymbol{J}_e^+ \boldsymbol{J}_e)\big)^+ (\dot{\boldsymbol{x}}_0 - \boldsymbol{J}_0 \boldsymbol{J}_e^+ \dot{\boldsymbol{x}}_e)$

8 $\quad \dot{\boldsymbol{q}} = \text{ClampAbsMax}(\dot{\boldsymbol{q}},\ 1)$

9 $\quad \boldsymbol{q} = \boldsymbol{q} + \dot{\boldsymbol{q}} \Delta t$ ▷ configuration is being updated

end

5 6D Object Pose Estimation

Mathias Emil Slettemark-Nielsen

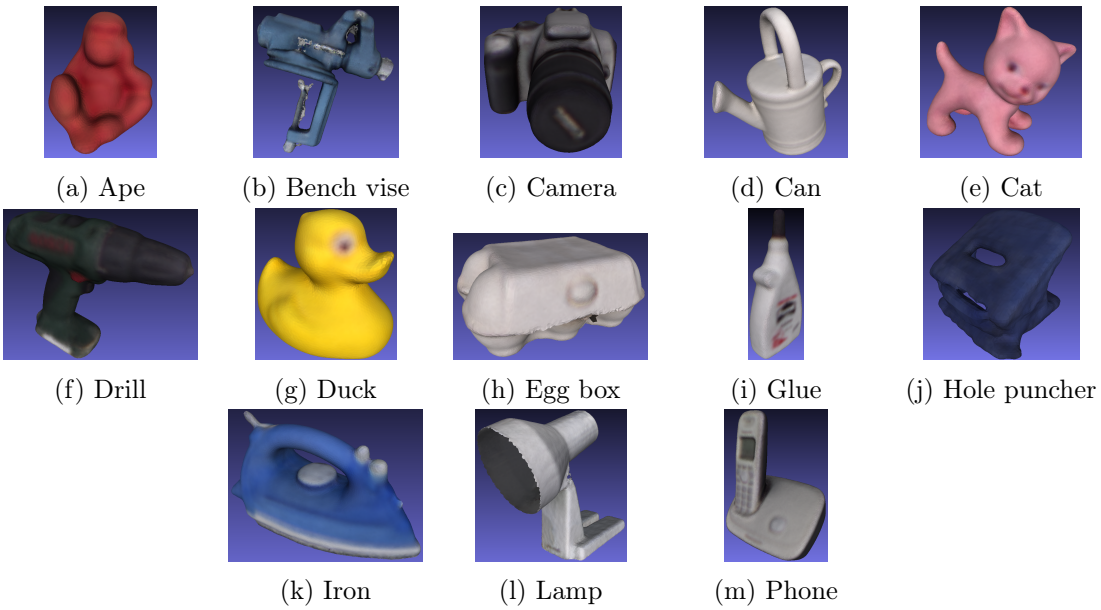
Since the previous sections assume that the pose of the obstacles are known, it is relevant to inspect methods that can be used to recognize and pose estimate a potential obstacle. In this section the dataset LINEMOD [9] will be used to determine the performance of the pose estimating network iterative dense fusion and compared against a 3D-3D pose estimating method.

5.1 The Dataset of Objects

Mikkel Larsen

Before delving into pose estimation a dataset is introduced. The dataset should contain variations in scale, viewpoint and illumination. Furthermore, the dataset should contain cluttered scenes and occlusion of objects.

The dataset used in this report is the LINEMOD dataset and can be found here [10]. The LINEMOD dataset consists of 13 objects which is shown in figure 29. The dataset contains a 3D model of each object, the maximum diameter of each object, the color images, the aligned depth images and the ground truth mask, rotation and translation.



(a) Ape (b) Bench vise (c) Camera (d) Can (e) Cat

(f) Drill (g) Duck (h) Egg box (i) Glue (j) Hole puncher

(k) Iron (l) Lamp (m) Phone

Figure 29: Illustrations of the objects from the LINEMOD dataset.

Examples of the data from the LINEMOD dataset are shown in figure 30. The examples illustrates the variantions in scale, viewpoint and illumination. Moreover, the cluttered scenes and the occlusion of objects can be seen.

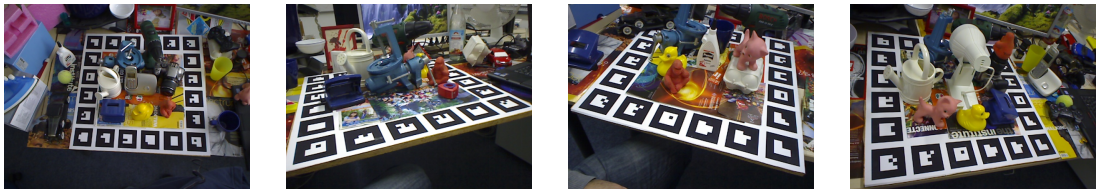


Figure 30: Examples of data from the LINEMOD dataset.

The LINEMOD dataset consist of 1236 ape images, 1215 bench vise images, 1201 camera images, 1196 can images, 1179 cat images, 1188 drill images, 1254 duck images, 1253 egg box images, 1220 glue images, 1237 hole puncher images, 1152 iron images, 1227 lamp images and 1225 phone images. Thus, a total of 15783 images.

To evaluate pose estimations the matching scores ADD, for non-symmetric object, and ADD-S, for symmetric objects are used, which is introduced in [11]. The symmetric objects are the egg box shown in figure 29h and the glue shown in figure 29i. The average ADD matching score for a 3D model $\mathcal{M}$ is computed as

$$\text{ADD score} = \underset{x \in \mathcal{M}}{\text{average}} \left\| (Rx + T) - (\tilde{R}x + \tilde{T}) \right\| \tag{52}$$

where the ground truth is denoted R for rotation and T for translation, $\tilde{R}$ and $\tilde{T}$ are the estimated rotation and translation, respectively, and x is the model points. The average ADD-S matching score is computed as

$$\text{ADD-S score} = \underset{x_1 \in \mathcal{M}}{\text{average}} \min_{x_2 \in \mathcal{M}} \left\| (Rx_1 + T) - (\tilde{R}x_2 + \tilde{T}) \right\| \tag{53}$$

where x_2 is closest point to x_1 within $\mathcal{M}$.

To classify if a pose is correct compared to the ground truth pose, Equation 54 must be true, else is the estimated pose classified as a wrong pose.

$$k_m \cdot d \geq m \tag{54}$$

Where k_m is a chosen constant which is set to 0.1 and d is the diameter of $\mathcal{M}$.

5.2 Iterative Dense Fusion

`Mathias Emil Slettemark-Nielsen`

To estimate a 6D object pose the method Iterative Dense Fusion, DenseFusion, is implemented. DenseFusion is introduced in [12] and uses deep networks for pose estimation from RBG-D inputs. Figure 31 illustrates an overview of DenseFusion.

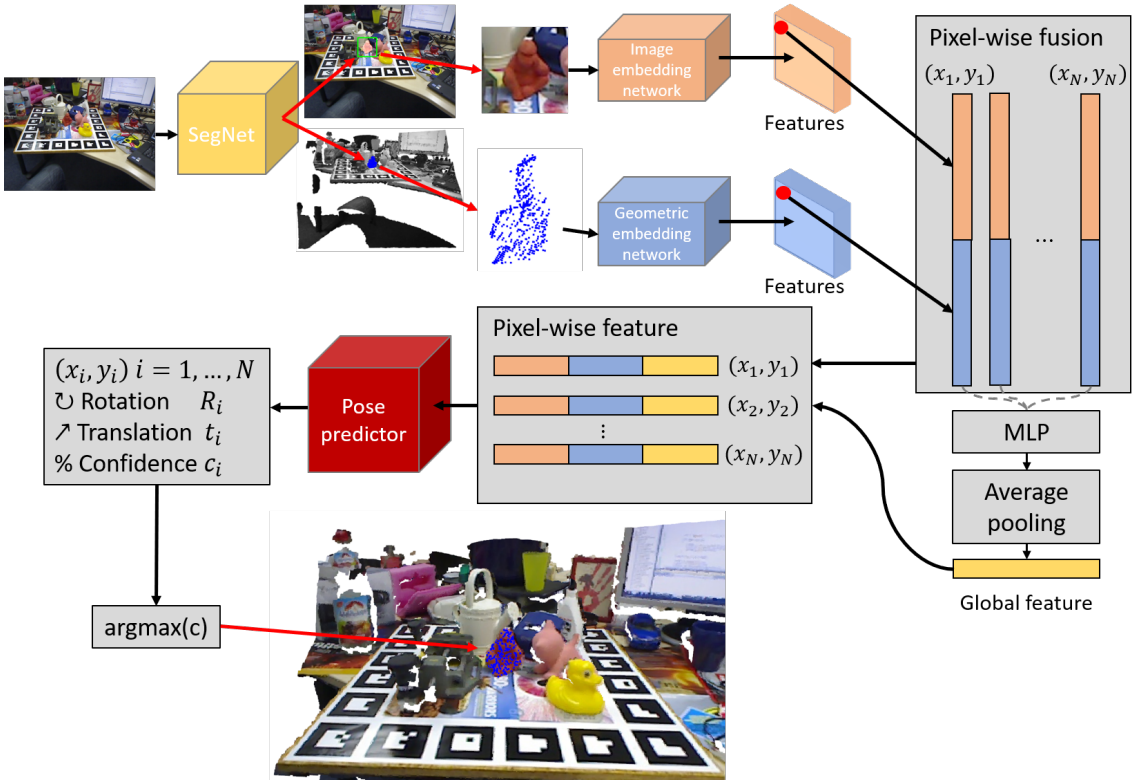


Figure 31: An illustration of the 6D pose estimation method DenseFusion, where the object of interest is found in the RGB-D input. The RGB-D input is then cropped and the RGB and D input is fed through different networks, thus creating color and point features. The features are fused with a global featue, and a network predicts a pose for each feature. The pose with the highest confidence score is the final 6D pose estimation.

DenseFusion consist of five stages:

1. A deep network that performs semantic segmentation, i.e. linking each pixel to a class label. The found mask of the object is used to create a point cloud of the object and a cropped color image.
2. A fully convolutional network that processes the color information and maps each pixel in the cropped image to a color feature vector, thus creating a feature map.
3. A PointNet-based [13] network that processes each point in the masked 3D point cloud to a geometric feature vector, thus creating a feature cloud.
4. A pixel-wise fusion network that combines color feature and geometric feature with a global feature, and outputs a estimation of the 6D pose for each fusion. A confidence is estimated for each fusion to pick the final pose.
5. A network to improve the predicted pose by estimating a residual pose.

5.2.1 Semantic Segmentation

Mathias Emil Slettemark-Nielsen

To be able to estimate the pose of an object, the object must first be identified in the image. This is done by the use of the deep network SegNet [14] which is created for semantic segmentation. Other alternatives such as U-Net [15] could also have been used for semantic segmentation. SegNet consist of an encoder network, a corresponding decoder network and a pixel-wise classification layer. The architecture of SegNet is illustrated in figure 32.

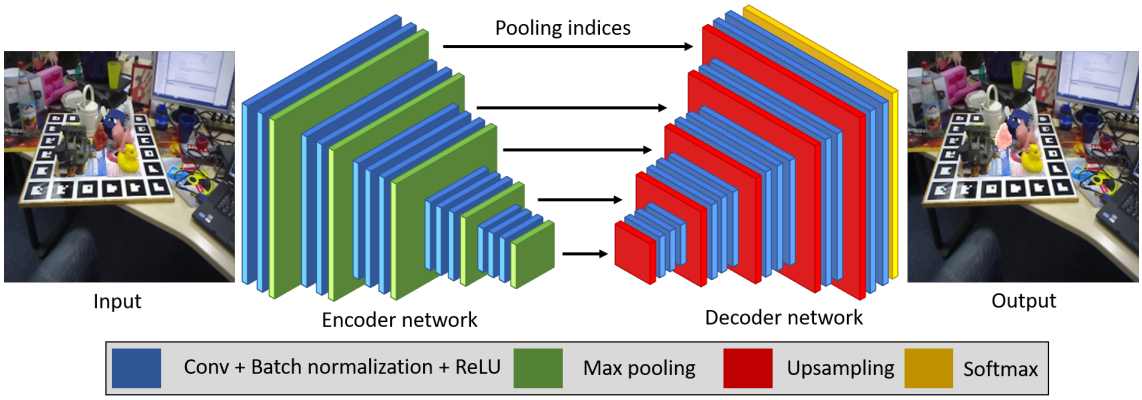


Figure 32: An illustration of the SegNet architecture, where the input image is from the LINEMOD dataset, and the output is the input with the prediction added.

Encoder network: To find features in the input image the first 13 convolutional layers in the VGG16 network [16] is used. The encoder network consist of several encoders where each encoder performs convolution, batch normalization [17] and Rectified Linear Unit, ReLU, [18]. After each encoder max pooling with a (2×2) window and a stride of 2 is performed. The locations of the maximum feature value in each max pooling window, the pooling indices, is stored for each encoder feature map.

Decoder network: To upsample the feature map the decoder network consist of a hierarchy of decoders, one for each corresponding encoder. Each decoder uses the corresponding pooling indices to upsample the input feature map. This is illustrated in figure 33, and it can be seen that a sparse feature map is produced. After each upsampling convolution, batch normalization and ReLU is performed on the feature map.

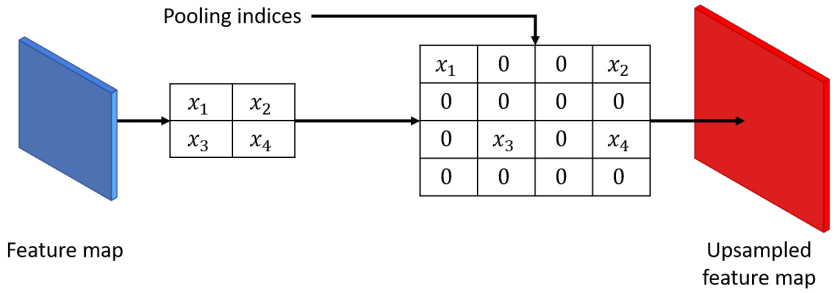


Figure 33: An illustration of SegNet decoders where x_1, x_2, x_3 and x_4 correspond to values in a feature map. The pooling indices are used to upsample the feature map.

Pixel-wise classification: The final feature map is fed to a softmax classifier [19], that classifies each pixel independently. The output of the softmax classifier is a image of probabilities with as many channels as number of classes, e.g. background and ape.

Implementation: The LINEMOD dataset provides masks with one object, thus a SegNet model has to be trained for each object. The following models from figure 29 are implemented: ape, bench vise, camera, cat and duck. The dataset is split into approximately 15 % training and 85 % testing, where the training is repeated for 20 iteration and then the model is tested on the testing set.

The training set is artificially enlarged by the use of data augmentation, where rotation and color jitter is applied. For rotation augmentation the data is either flipped left to right, up to down or left to right followed by up to down. For color augmentation the data is jittered in brightness, contrast, saturation and hue.

The Adam optimizer [20] is used with a learning rate of 0.0001, coefficients for computing running average of gradient $\beta_1 = 0.9$ and its square $\beta_2 = 0.999$ and the term added to the denominator to improve numerical stability $\epsilon = 10^{-8}$.

Evaluation: To evaluate the trained SegNet model the SegNet results from [21] are used which can be found here [22]. For evaluation of object detection the metric Intersection over Union, IoU, is used and is illustrated in figure 34.

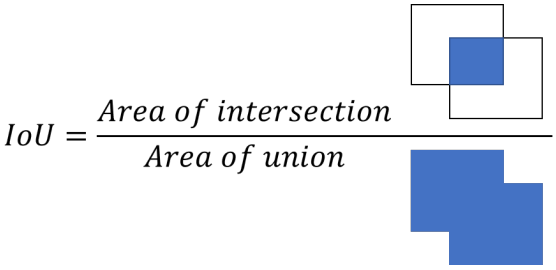


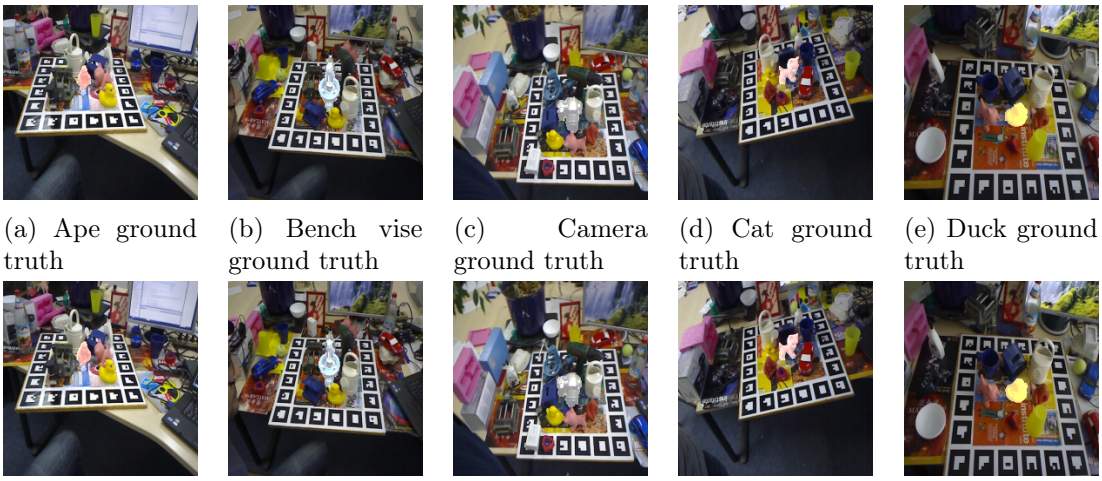
Figure 34: An illustration of Intersection over Union, IoU. The intersection is the overlap of the prediction and ground truth, and the union is the combined area.

The results of the trained SegNet models is shown in table 1, where the amount of data, average accuracy, average IoU and the average prediction time can be seen.

	Ape	**Bench vise**	**Camera**	**Cat**	**Duck**
Data	1050	1031	1020	1002	1065
Accuracy [%]	99.9	99.6	99.7	99.9	99.9
IoU	0.908 (±0.052)	0.865 (±0.046)	0.8839 (±0.0355)	0.876 (±0.032)	0.911 (±0.044)
Time [ms]	0.0039 (±0.0008)	0.0038 (±0.0006)	0.0038 (±0.0006)	0.0037 (±0.0006)	0.0039 (±0.0006)

Table 1: Results of the implemented SegNet models.

From table 1, the accuracy of each of the SegNet models are close to perfect. Furthermore, the lowest IoU score is 0.865 and a good IoU score is deemed to be above 0.7. Examples of the trained SegNets are shown in figure 35.



(a) Ape ground truth (b) Bench vise ground truth (c) Camera ground truth (d) Cat ground truth (e) Duck ground truth

(f) Ape prediction (g) Bench vise prediction (h) Camera prediction (i) Cat prediction (j) Duck prediction

Figure 35: Examples of semantic segmentation of the LINEMOD dataset. The first row contain the ground truth labels added to the input image, and the second row the corresponding predictions also added to the input image.

The examples in figure 35, shows that the SegNet models detects each pixel of the object. The generated mask from the SegNet can now be used for feature extraction.

5.2.2 Feature Extraction

`Mathias Emil Slettemark-Nielsen`

In DenseFusion the RGB input and depth input are processes separately to generate color and geometric features, because the RGB and depth information resides in different spaces. Therefore, two different network is implemented one for generate color embedding features and one for geometric embedding features.

3D point cloud features: The segmented depth pixels are converted into a 3D point cloud using the known camera intrinsics. The 3D point cloud is then fed to a geometric embedding network, that for each point generates a feature by mapping points to a $d_{geometric}$-dimensional feature space. This network is referred to as the geometric embedding network in figure 31. The architecture is a multi-layer perceptron, MLP, followed by average pooling.

Color image features: To extract image features a CNN-based encoder-decoder architecture that maps an image of size $(H \times W \times 3)$ into $(H \times W \times d_{rgb})$ space is used. Thereby, making each pixel a d_{rgb}-dimensional vector that represents the input image pixel. This network is referred to as the image embedding network in figure 31. The network consists of a ResNet-18 [23] encoder followed by four upsampling layers as the decoder, the architecture of the network is illustrated in figure 36.

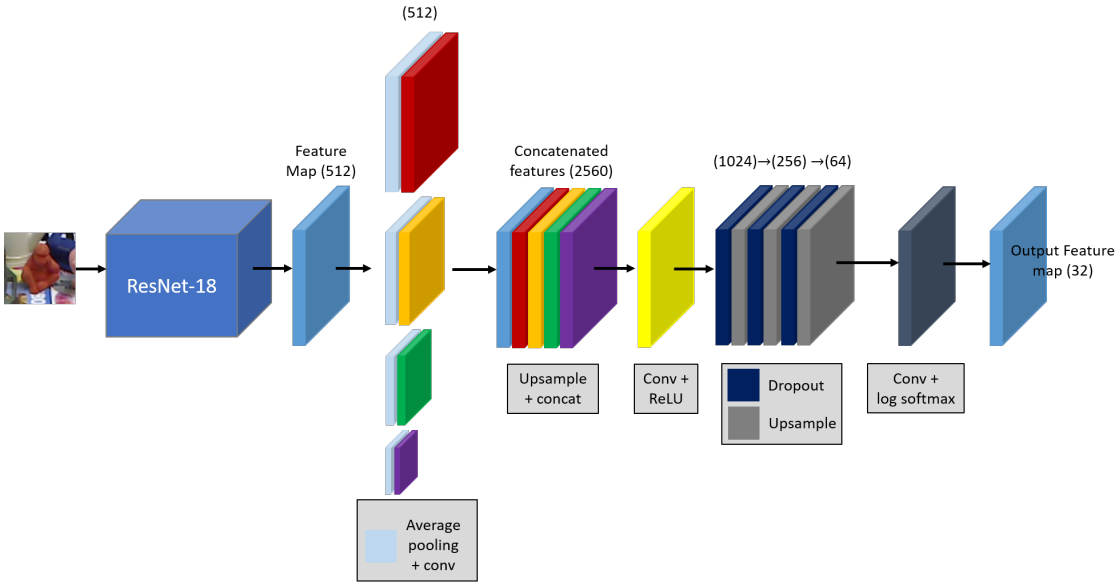


Figure 36: An illustration of the image embedding network that encodes and decodes a image.

The image embedding network in figure 36 is inspired by Pyramid Scene Parsing Network, PSPNet, from [24]. PSPNet performs scene parsing based on semantic segmentation, thus it predicts the label, location and shape for each object. PSPNet introduces a pyramid pooling module that fuses features under four different scales, thereby making the detection more robust. The final output is a feature map that is passed on to the pose predictor network along with the gemometric featues as illustrated in figure 37.

5.2.3 Pixel-wise Fusion for Pose Prediction

`Mathias Emil Slettemark-Nielsen`

The extracted features from the color image and 3D point cloud is now fused, thus each point contain information for each domain. DenseFusion performs local per-pixel fusion instead of global fusion so that predictions can be made based on each fused feature.

The corresponding geometric feature and color feature are found based on a projection onto the image plane using the known camera intrinsic parameters. These features are then concatenated and used to generate a global feature, by a MLP followed by a averaging pooling. The fused feature is concatenated with the global feature and fed to a pose predictor network that predicts the 6D pose of the object, this process is illustrated in figure 37.

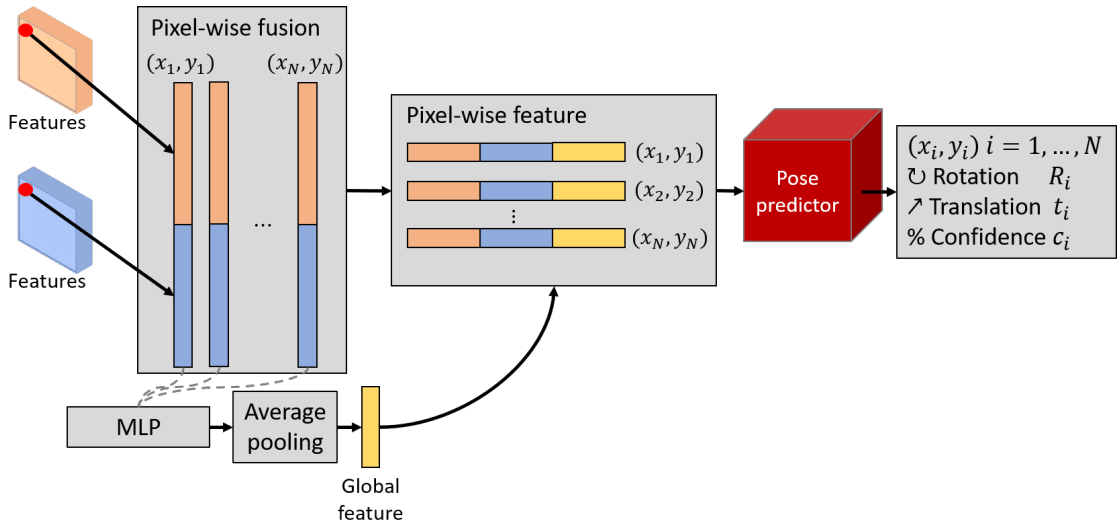


Figure 37: An illustration of DenseFusion pose prediction where an estimated pose is generated for each pixel feature.

The pose predictor network consists of three in parallel networks for rotation, translation and confidence. The networks consists of three MLP's with ReLU as activation function and a fourth MLP which differs for all three. The fourth MLP for the rotation has an output of four which represents a quaternion, likewise the fourth MLP for the translation

has an output of three which represents the position in space. The fourth MLP for the confidence only has one output, the level of confidence for the estimated pose, and uses the sigmoid function as activation function.

The pose predictor network is trained to predict a pose to each fused feature. To decide the best estimated pose, the pose predictor outputs a confidence score c_i for each pose estimation. This gives the pose predictor network two learning objective, pose estimation and confidence score. The pose estimation loss is the distance between the point from the ground truth pose and the corresponding point from the estimated pose. The pose estimation loss function to minimize is

$$L_i^p = \frac{1}{\mathcal{M}} \sum_{j=1}^{\mathcal{M}} \left\| (Rx_j + T) - (\tilde{R}_i x_j + \tilde{t}_i) \right\| \tag{55}$$

where x_1 is points of the objects 3D model $\mathcal{M}$, $p = [R|t]$ is the ground truth pose, and $\tilde{p}_i = [\tilde{R}_i|\tilde{t}_i]$ is the estimated pose generated from the fused feature of the i'th pixel. However, this loss function is only well defined for non-symmetric objects. Thus, for symmetric objects, the loss function finds the closest point between estimated model and the ground truth model. The loss function for symmetric objects is

$$L_i^p = \frac{1}{\mathcal{M}} \sum_{j=1}^{\mathcal{M}} \min_{k \in \mathcal{M}} \left\| (Rx_j + T) - (\tilde{R}_i x_k + \tilde{t}_i) \right\| \tag{56}$$

To learn the network to balance confidence among the predictions, the per pixel pose estimation loss is weighted with the corresponding confidence and a confidence regularization term is added. Thereby, the overall loss function is calculated as

$$L = \frac{1}{N} \sum_{i=1}^{N} \left(L_i^p c_i - w \log(c_i) \right) \tag{57}$$

where w is a balancing hyperparameter that is set to 0.015 as in the paper [12], and N denotes the number of points randomly sampled from the segment. The pose estimation with the highest confidence is the final prediction as shown in figure 31.

5.2.4 Iterative Refinement

`Mathias Emil Slettemark-Nielsen`

DenseFusion uses an iterative refinement method to improve the predicted pose. This is done instead of using the Iterative Closest Point, ICP, algorithm [25]. The pipeline of the iterative refinement method is illustrated in figure 38.

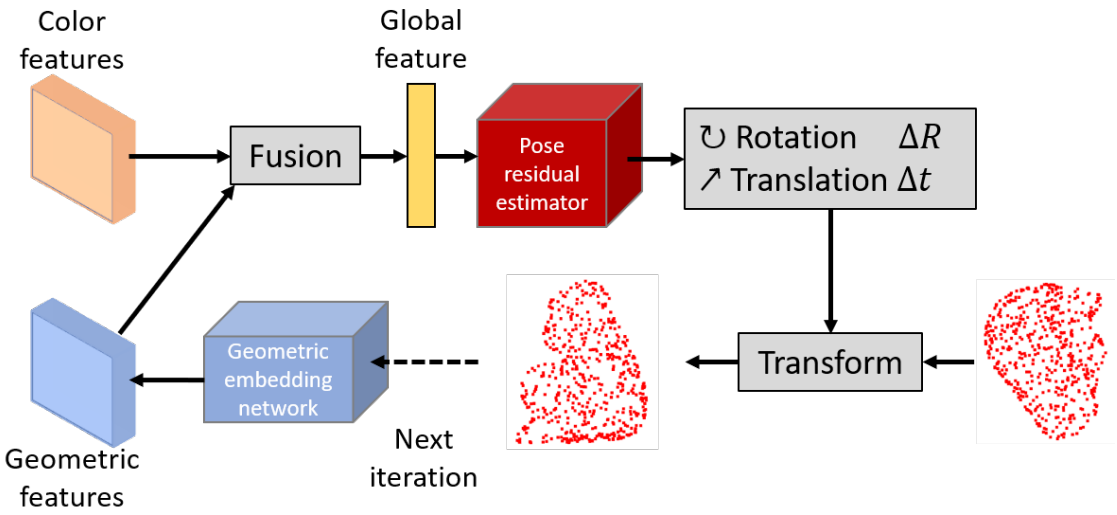
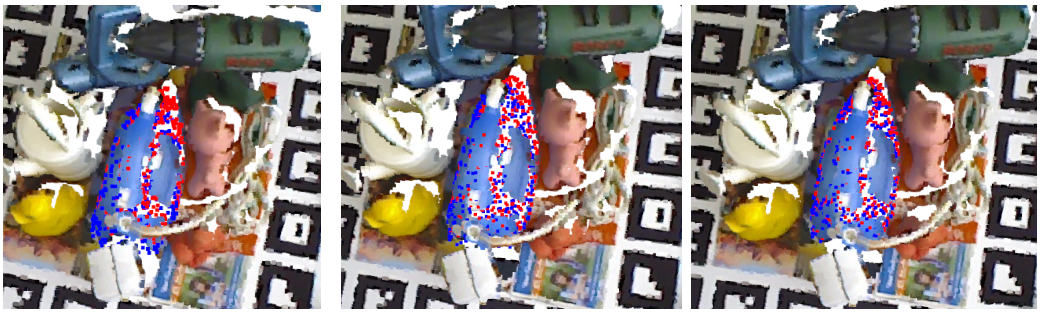


Figure 38: Iterative refinement of the predicted pose where a pose residual estimator predicts a residual pose.

The iterative refinement network takes the latest estimated pose and feeds it into the geometric embedding network. The color features and the new geometric features are fused to global features, which is fed to the pose residual estimator. The pose residual estimator predicts a residual pose based on the previously estimated pose and the generate global feature. After a number of iterations, k, the final pose estimation is obtained as

$$\tilde{p} = [R_k|t_k] \cdot [R_{k-1}|t_{k-1}] \cdots [R_0|t_0] \tag{58}$$

The iterative refinement network consist of four fully connected layers that output the pose residual from the global feature, as illustrated in figure 38. An illustration of the iterative refinement applied to a pose prediction is shown in figure 39, where the reduction in the ADD score can be seen.



(a) ADD score: 0.0229 (b) ADD score: 0.0131 (c) ADD score: 0.00581

Figure 39: Illustration of the iterative refinement procedure, where the estimate pose (blue points) and the ground truth pose (red points) is shown.

In figure 39, it can be seen that the estimated pose, the blue points, gets closer to the ground truth pose, the red points, for each refinement iteration. This can also be seen by looking at the calculated ADD score that is descending. Further visualization of the iterative refinement procedure can be found at GitHub[4].

5.2.5 Implementation of Iterative Dense Fusion

Mathias Emil Slettemark-Nielsen

The LINEMOD dataset is split in approximately 15 % training and 85 % testing, as in the SegNet in section 5.2.1. The training set is repeated 20 times before testing, and augmentation is added to the input image and the translation part of the 3D point cloud. The image augmentation is jittering in brightness, contrast, saturation and hue, and the 3D point cloud augmentation is random noise between ± 0.03.

In the paper [12], they recommend to start the training of the iterative refinement network after the pose predictor has converged. Thus, the training is set to start when the pose predictor has a loss under 0.013. The number of iterations, k, is set two 2 as in the paper [12]. A learning rate of 0.0001 is used with a decay of 0.3, which is used when the network has a loss under 0.016.

DenseFusion requires a lot of training and because of limited time and resources, the trained checkpoints from [12] is used as start parameters for the network.

5.2.6 Evaluation of Iterative Dense Fusion

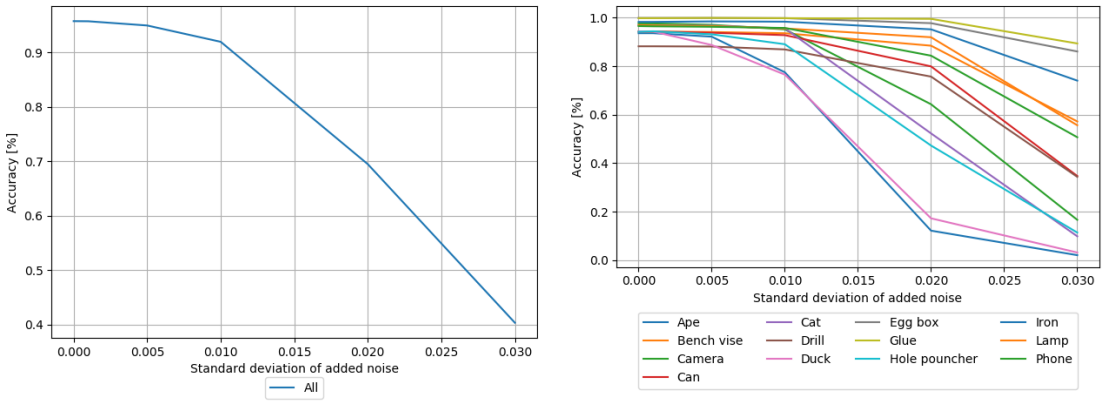
Mathias Emil Slettemark-Nielsen

Table 3 shows the results of the DenseFusion model on the LINEMOD dataset, and the evaluation as in [11] as described in section 5.1 with the ADD/ADD-S metric. The prediction time of the DenseFusion method is $0.0259(\pm 0.0156)$ seconds. Visualization of the pose estimation by DenseFusion can be seen on GitHub[5].

To evaluate the robustness of the DenseFusion model noise is added to the input, both the RGB image and the 3D point cloud. The noise added is sampled from a normal distribution with zero mean and varying standard deviation, and the noise added to the RGB image is scaled with the maximum pixel value 255. The noise experiments can be seen in figure 40.

[4] https://github.com/Masle16/obstacle_avoidance_with_dmps/blob/master/pose_estimation/dense_fusion/REFINENET.md

[5] https://github.com/Masle16/obstacle_avoidance_with_dmps/blob/master/pose_estimation/dense_fusion/RESULTS.md



(a) The overall accuracy when noise is added to the input.

(b) Accuracy of each object when noise is added to the input.

Figure 40: Noise experiments of DenseFusion.

From figure 40a, the overall accuracy of DenseFusion drops below 50 % when noise with a standard deviation of 0.03 is applied. However, when looking at the individual objects in 40b some objects still have a accuracy above 50 %. The object Ape and Cat drops below 50 % at a standard deviation of 0.015. The performance of DenseFusion is deemed acceptable until a standard deviation of 0.01 where the overall accuracy is 91.98 %, and the lowest individual accuracy is 77.5 %.

5.3 3D - 3D Pose Estimation

`Mikkel Larsen`

In order to compare the performance of the 6D pose estimation of the DenseFusion network a baseline method is chosen. The method is a traditional pose estimation method consisting first of a global pose alignment followed by a local pose alignment. The input to the method is a 3D model of the object and a 3D scene where the object has been segmented. Both the 3D model and scene is a point cloud. The 3D segmented object is generated by the SegNet as described in subsubsection 5.2.1. The implemented global alignment and local alignment method used is from the library open3d [26].

5.3.1 Alignment Parameter Choice

The global alignment is a RANSAC algorithm using both a point cloud of the model and the segmented object and corresponding point cloud feature descriptors. The implemented global alignment method uses Fast Point Feature Histogram, FPFH, as a descriptor. The FPFH uses surface normal for each point to construct a k-d tree in order to related the points to each other. Here the radius for the surface normal is set to 0.25 and the maximum number of neighbor points to 15. The k-d tree parameters are set to a radius feature of

0.5 and maximum number of neighbor point of 30.

The RANdom SAmple Consensus algorithm, RANSAC, saves the best transformation based on point inliers from model point cloud to segmented point cloud. The distance threshold between points from the model and segmented is set to 0.1. The same distance is also used when specifying the maximum distance between feature point pairs. Furthermore, the maximum number of iterations of RANSAC is either $4 \cdot 10^5$ or 10^4 new best transformations. Using 10^4 yielded significant better accuracy but also longer evalution time as shown in Table 2.

The local alignment method uses Iterative Closest Point, ICP. The same distance threshold between points from the model point cloud and the segmented point cloud of 0.1 is used.

5.3.2 Evaluation of 3D - 3D Pose Estimation

A test to decide the maximum number of validation were conducted. Because of time constraint the test were only conducted on the first 10 objects of each class of the LINEMOD dataset. The results is listed in Table 2.

Max number of validations	10^4	10^3	10^2	10
Total Accuracy [%]	62.3	47.7	22.3	19.2
Iterations pr time [it/s]	0.58	3.77	7.83	8.95

Table 2: Test of max number of validation of RANSAC in 3D-3D pose alignment, logging time and accuracy. Tested with all object classes but only for the first 10 objects.

The same evaluation is done for 3D-3D pose estimation as for DenseFusion described in subsubsection 5.2.6 on the LINEMOD dataset. The results is shown in Table 3.

	DenseFusion		3D - 3D	
	Accuracy [%]	ADD/ADD-S [cm]	Accuracy [%]	ADD/ADD-S [cm]
Ape	93.71	0.005(±0.00603)	51.38	0.02457(±0.02217)
Bench vise	94.67	0.00767(±0.0146)	59.94	0.03895(±0.04337)
Can	97.45	0.00638(±0.00928)	44.61	0.04761(±0.04139)
Camera	94.69	0.00675(±0.0143)	52.17	0.04602(±0.04364)
Cat	96.71	0.00468(±0.00852)	57.78	0.02940(±0.03124)
Drill	88.31	0.0143(±0.0263)	65.01	0.03546(±0.04351)
Duck	93.90	0.00629(±0.0191)	46.49	0.02989(±0.03083)
Eggbox	99.81	0.00763(±0.0011)	99.81	0.00744(±0.002588)
Glue	99.81	0.00481(±0.00192)	99.90	0.00611(±0.002708)
Hole pouncher	94.01	0.00683(±0.0102)	37.68	0.04104(±0.03168)
Iron	98.47	0.00651(±0.0111)	41.47	0.07298(±0.05758)
Lamp	97.02	0.00638(±0.0143)	66.41	0.03337(±0.04200)
Phone	96.35	0.00693(±0.0121)	73.97	0.02137(±0.02907)
All	95.76	0.00693(±0.0133)	61.28	0.03510(±0.03240)

Table 3: Result of the implemented DenseFusion and 3D-3D pose estimation methods on the LINEMOD dataset.

From table 3, the method DenseFusion scores a overall accuracy of 95.76 % whereas the 3D-3D gets 61.28 %. The 3D-3D method only gets a better accuracy with the glue object. However, there is only a difference between the two methods of 0.09. The prediction time of DenseFusion is 0.0259 seconds and 3D-3D is 1.72 seconds, thus DenseFusion predicts significantly faster than the 3D-3D method.

6 Discussion

The tuning process of potential fields for obstacle avoidance in DMPs is a time consuming process and must be repeated for each type of obstacle including shape and size. To avoid this time consuming and individual process another demonstration could be presented, where a human demonstrates how to avoid an obstacle. By doing this and adding an extra term to the DMP, it might be possible to fit the DMP to the demonstration and thus learn parameters for the potential fields instead of tuning them by hand.

The damping term, λ, in section 4 was chosen empirically, but can be chosen dynamically [8], which might show to be beneficial for other trajectories and obstacles than those used.

The implementation of link collision avoidance does not consider joint accelerations, thus if the link collision avoidance was to be implemented in a real system, it might show to be necessary to inspect joint accelerations and incorporate a limit on these.

The pose estimation methods uses the LINEMOD dataset described in section 5.1 for performance evaluation and the objects can be seen in figure 29. The objects are fairly simple to distinguish between, since most of the objects have different color and very different shape. Moreover, the objects of interest are mainly placed in the center of the images as illustrated in figure 30. Therefore, for future work the implemented methods should also be evaluated on other datasets or a self made dataset.

At the start of the project the training of DenseFusion from section 5.2 was done from scratch. However, after several training hours with small growth in accuracy, it was decided to used the pretrained DenseFusion model form [22]. Thus, for future work the full training time of DenseFusion should be estimated.

In section 5, two pose estimation methods are compared, DenseFusion and 3D-3D pose estimation. It was shown that DenseFusion outperforms the 3D-3D pose estimation on the LINEMOD dataset. However, the implementation and training of DenseFusion is far more complicated than 3D-3D pose estimation. Therefore, when choosing between the two methods the implementation time must be taken into account.

7 Conclusion

Two repulsive fields for end-effector obstacle avoidance incorporated into dynamic movement primitives were designed and evaluated. It was shown that a repulsive term based on coupled terms with addition of an attractional field towards the demonstrated trajectory had the best performance with respect to avoiding obstacles and still following the shown demonstration, but at the expense of slight oscillation around the demonstrated path, when avoiding an obstacle.

A link collision avoidance algorithm for a 7-DOF robot manipulator was designed that was able to follow a trajectory with the end-effector while avoiding collisions between obstacles and the links of the robot. The link collision algorithm is based on damped least squares, null space movement based on a potential field around the obstacles and a clamping function limiting the generated joint velocities to improve the response when operating close to singularities.

The 6-DOF pose estimation method DenseFusion was implemented and evaluated on the LINEMOD dataset. DenseFusion acheives an overall accuracy of 95.76 % on the LINEMOD dataset. The implemented DenseFusion method outperforms a standard 3D-3D pose estimation.

References

[1] Auke Jan Ijspeert et al. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". en. In: *Neural Computation* 25.2 (Feb. 2013), pp. 328–373. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/NECO_a_00393. URL: http://www.mitpressjournals.org/doi/10.1162/NECO_a_00393 (visited on 04/13/2020).

[2] Iñigo Iturrate San Juan. "Handed DMP template". In: (2020). URL: https://e-learn.sdu.dk/bbcswebdav/pid-5861069-dt-content-rid-10911304_2/xid-10911304_2.

[3] Ales Ude et al. "Orientation in Cartesian space dynamic movement primitives". en. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 2997–3004. ISBN: 978-1-4799-3685-4. DOI: 10.1109/ICRA.2014.6907291. URL: http://ieeexplore.ieee.org/document/6907291/ (visited on 05/12/2020).

[4] W. Warren, B. Fajen, and D. Belcher. "Behavioral dynamics of steering, obstacle avoidance, and route selection". en. In: *Journal of Vision* 1.3 (Mar. 2010), pp. 184–184. ISSN: 1534-7362. DOI: 10.1167/1.3.184. URL: http://jov.arvojournals.org/Article.aspx?doi=10.1167/1.3.184 (visited on 04/13/2020).

[5] Akshara Rai et al. "Learning coupling terms for obstacle avoidance". en. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE, Nov. 2014, pp. 512–518. ISBN: 978-1-4799-7174-9. DOI: 10.1109/HUMANOIDS.2014.7041410. URL: http://ieeexplore.ieee.org/document/7041410/ (visited on 04/13/2020).

[6] Anthony A. Maciejewski and Charles A. Klein. "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments". en. In: *The International Journal of Robotics Research* 4.3 (Sept. 1985), pp. 109–117. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836498500400308. URL: http://journals.sagepub.com/doi/10.1177/027836498500400308 (visited on 04/23/2020).

[7] Dae-Hyung Park et al. "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields". en. In: *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*. Daejeon: IEEE, Dec. 2008, pp. 91–98. ISBN: 978-1-4244-2821-2. DOI: 10.1109/ICHR.2008.4755937. URL: http://ieeexplore.ieee.org/document/4755937/ (visited on 04/20/2020).

[8] Samuel R Buss. "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods". en. In: (), p. 19.

[9] S. Hinterstoisser et al. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes". In: *2011 International Conference on Computer Vision*. 2011, pp. 858–865.

[10] Stefan Hinterstoisser. 2017. URL: http://campar.in.tum.de/Main/StefanHinterstoisser (visited on 04/21/2020).

[11] Stefan Hinterstoisser et al. "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes". In: *Asian conference on computer vision*. Springer. 2012, pp. 548–562.

[12] Chen Wang et al. "DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion". In: (2019).

[13] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[16] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[17] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[18] *CS231n, Commonly used activation functions*. https://cs231n.github.io/neural-networks-1/. Accessed: 2020-05-13.

[19] *Convolutional Neural Networks for Visual Recognition*. https://cs231n.github.io/linear-classify/. Accessed: 2020-05-12.

[20] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Yu Xiang et al. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes". In: *arXiv preprint arXiv:1711.00199* (2017).

[22] *DenseFusion LINEMOD Preprocessed*. https://drive.google.com/drive/folders/19ivHpaKm9dOrr12fzC8IDFczWRPFxho7. Accessed: 2020-05-01.

[23] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[24] Hengshuang Zhao et al. “Pyramid Scene Parsing Network”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[25] Paul J Besl and Neil D McKay. “Method for registration of 3-D shapes”. In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.

[26] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018).