Vision-guided object pose estimation for robotic pushing in real-time

P.M. van der Burg

Delft University of Technology

Vision-guided object pose estimation for robotic pushing in real-time

by

P.M. van der Burg

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended on August 7 2023.

Student number:	4292286	
Thesis committee:	Prof. dr. ir. M. Wisse	TU Delft, supervisor
	Ir. Y. Tang	TU Delft, daily supervisor
	Dr. ir. H. Caesar	TU Delft, external committee member
Project Duration:	June, 2022 - August, 2023	
Faculty:	Faculty of Mechanical Engineering	
	Department of Cognitive Robotics	
	Delft University of Technology	

Vision-guided object pose estimation for robotic pushing in real-time

P.M. van der Burg

Abstract—Object pushing in robotics has numerous applications, but it often relies on room-bound object tracking systems such as Motion Capture (MoCap) for accurate object pose acquisition. Such systems limit the potential use scenarios, since they add complexity and cost and require expansion of the sensor infrastructure for expanding the operational areas. To address these limitations, we propose a framework that integrates object pushing and vision-guided object pose estimation using a monocular RGB camera that is placed on the pushing agent. By leveraging the temporal coherence of the scene, we accelerate the 2D object detection stage that is common in the field of object pose estimation. Our contributions include the development of a novel framework that continuously pushes an object while simultaneously tracking its pose. We incorporate dynamics by efficiently using a Model Predictive Path Integral controller's predictive model to predict the object location in the forthcoming image, resulting in an extreme reduction of computational cost for 2D object detection, without sacrificing the accuracy of the pose estimation. Our framework achieves real-time object pushing with accurate object pose estimation, which we demonstrate with real-world experiments. Furthermore, we provide a novel object pose estimation dataset in the widely used BOP format, specifically for robot planar pushing with an on-board camera. Our approach pushes towards eliminating the need for room-bound sensor systems, expanding the potential use cases. By considering temporal coherence and scene dynamics, our framework challenges recent object pose estimation methods that fully process each image as uncorrelated individuals, while providing a promising solution for real-time object pushing.

Index Terms—object pose estimation, computer vision, deep learning, robot pushing, non-prehensile manipulation

1 Introduction

In this work we consider the estimation of an object's position and orientation using computer vision in the context of non-prehensile robotic manipulation, specifically pushing. Robot object pushing has recently gained popularity in the field of robotics in the context of applications such as industrial automation [38] and autonomous navigation [17]. Because of the friction variability and stochastic nature of object pushing [51], the object might slip or unexpectedly turn away from the robot's end-effector during the pushing action. The controller must be able to quickly respond to these changes, thus it is crucial to have precise and uninterrupted knowledge of the object's pose (position and orientation). This information is typically obtained using a room-bound sensor system, such as a Motion Capture (MoCap) system with infrared technology and reflective markers placed on the robot and object(s) [3][15], since it is accurate and fast. However, these location-specific systems introduce complexity, as they require regular time-consuming and labour intensive calibration [30], during which the system is non-operational. Moreover, expanding the operation space necessitates a proportional growth in the sensor infrastructure with accompanying cost and increasing calibration complexity.

Therefore, we propose to acquire the object's pose using computer vision. An RGB camera is placed on the pushing agent and we deploy an object pose estimation algorithm that analyses the RGB image and outputs an object pose estimate. Recent object pose estimation methods employ learning-based techniques trained on annotated data from RGB or RGB-D sensors to estimate the 6 Degrees of Freedom (6DoF) pose of the object. RGB-D methods generally achieve higher accuracy [9], since estimating depth and scale from an RGB image is a challenging problem [16]. Nevertheless, RGB-based methods still hold value as RGB sensors are widely available and relatively inexpensive.

To reduce the computational load and information delay associated with object pose estimation, we use an off-the-shelf object pose estimation method and accelerate the commonly used 2D detector stage by leveraging the temporal scene coherence and dynamics that are inherent in real-world robotic pushing scenarios. By using the predictive model from a Model Predictive Path Integral-based pushing controller, we can predict the object's position in the next image, allowing us to efficiently calculate the required 2D bounding box. To summarise, we make the following contributions:

- We introduce a novel framework for vision-guided object pushing using object pose estimation instead of MoCap.
- Our approach leverages temporal scene coherence, resulting in a reduction of the pose estimator's 2D detector runtime by over 50x, without sacrificing pose estimation accuracy.
- We evaluate the pushing performance of the proposed framework in real-world experiments.
- A new *AIRlab* dataset in the widely used BOP format [14] is introduced. It is focused on object pose estimation for non-holonomic planar pushing using an on-board RGB camera.

2 Related work

2.1 Object pose estimation

The field of object pose estimation has witnessed significant advancements in performance since the emergence of machine learning and Convolutional Neural Networks (CNNs) [20], [48], [52]. Early learning-based methods employed CNNs for direct regression or classification of rotation and translation [20], [48], but struggled with generalisation due to the non-linearity of the rotation space [52] and occlusions.

A two-stage approach was introduced to overcome the limitations of direct regression. In this approach, 2D keypoints are first detected using CNNs and then matched to an object's CAD model. The pose is then solved using Perspective-n-Point (PnP) [31], [35], [26]. Various techniques were incorporated to improve the 2D detection step, such as sliding window and probability heatmaps [31], semantic segmentation approaches [35], or utilizing off-the-shelf 2D detectors [26]. This two-stage approach, along with prior 2D detection, proved to be effective and became widely adopted in the field [32], [25], [47], [26], [24], [4]. However, these methods were limited by their reliance on matching only a few keypoints per object, making them sensitive to occlusions and truncations.

Recent advancements in pose estimation methods have focused on pose refinement approaches [25], [18] and matching pixel-wise correspondences [5], [32], [37], [39], [49]. These methods have significantly improved accuracy in the presence of occlusions or truncations. However, their speed is compromised due to increased computational load. The refinement step involves rendering the object model, which is computationally demanding. Dense correspondence matching requires processing every pixel, significantly increasing the number of calculations. Additionally, pose refinement methods rely on initial pose estimates obtained from models such as PoseCNN [48] or PVNet [33]. Similarly, dense correspondence methods assume the usage of a 2D detector such as FCOS [41] or Faster R-CNN [36]. These requirements add computational expenses and necessitate additional annotated training data for the additional models. Achieving practical solutions beyond benchmark performance, where initial pose estimates or 2D detections are readily available, becomes more challenging.

A common drawback of the aforementioned methods is the lack of consideration for temporal coherence in the target scene. These methods treat each image as an individual instance without considering the prior scene state and the dynamics of objects within the scene. Existing methods that do account for temporal coherence compromise on accuracy [9] or exclusively utilize RGB-D data [47]. Moreover, these methods rely on an initial pose estimate, which introduces errors that accumulate over time [11].

2.2 Locating the object in robotic pushing

In robot manipulation, the field predominantly covers prehensile manipulation of objects using fingers or other specialised appendages to grip or grasp an object. This allows for precise control and fine movements, making it easier to pick up, manipulate, and interact with objects in a more delicate and controlled manner. In this work, we consider pushing, which is considered non-prehensile manipulation, which relies on interactive contact that is not based on grasping. Recovering the pose of the object is only a concern before grasping in prehensile manipulation, since the object's pose and movements will be predictable after the grasp has been established. In non-prehensile manipulation, the requirement of accurate object pose information is continuous. Since we consider planar pushing, we limit the scope to methods that acquire the pose of the object continuously during planar pushing.

In [8], reinforcement learning-based semantic segmentation is used to infer object pose features, using images from an RGB camera that is mounted under the pushing plane, which has been made transparent to prevent occlusions. By using a Variational Auto-Encoder (VAE), they extract features from the image that are used for planning the control policy. The methods proposed in [23] and [29] both perform pushing using a holonomic base using AR-tags (fiducial markers) to locate the object in real-world experiments. The authors of [23] mount the camera on the robot, while [29] fixate the camera in the room. One disadvantage of fiducial markers is that by default, they must be placed on a flat surface, which complicates matters when curved objects are used. Moreover, fiducial markers require each object to be equipped with tags and calibrated so the centre of gravity relative to the tag is known by the pushing controller.

More recently, the authors of [50] propose an approach similar to our method, using a torque controlled robotic arm. The object's pose is estimated using a pre-trained DOPE framework [42], which is a deep learning-based 6DoF object pose estimation. To deal with the dynamics of the scene, they implement a particle filter [40] into the framework, which increases the computational load, but boosts the accuracy of the pose estimator and makes it more robust to occlusions. In this setup, the camera is fixated in the room while observing the scene. This may not be a large issue for a robot arm fixed to a table, however, with a driving robot, this severely limits the space of operation. Therefore we propose a method which performs object pose estimation using an on-board camera.

To the best of our knowledge, no previous work has addressed the problem of continuously estimating the pose of an object for object pushing with a non-holonomic robot using an on-board RGB sensor without fiducial markers.

3 METHOD

Our goal is to eliminate the MoCap system for robot pushing. Instead of using MoCap, we obtain the pose of the object using a computer vision-based object pose estimation method.

Pose estimation methods generally require a substantial time to calculate a pose. This limits the usability when continuously tracking an object in a real-time pushing scenario. Well-performing pose estimators commonly use a 2D detector to help the CNN 'focus' on a smaller image segment, which boosts accuracy [26]. A 2D detector takes the full-frame RGB image and returns the estimated bounding box around the target object. In ZebraPose [39], the authors report that the 2D detection step comprises 50% of the computation time of the entire pose estimation pipeline. We argue that this computation time can be reduced by leveraging the time coherence and dynamics of the scene. Therefore, we propose to substitute the vision-based detector with a physics-based implementation. We use the predictive dynamics model learned by and used in [34] to predict the future state of the object in the next image.

The control, prediction, detection and estimation modules are integrated into a single framework, which can be regarded a feedback loop in which each module is dependent on the previous module's output. We visualise this unified framework in Figure 1. This figure visually demonstrates the information flow of 1 full pipeline loop. Please note that the MoCap system is only used to supply the robot pose information, and we argue about its use in Section 5.

We divide the framework down into three main stages: (1) **Planning and Control**, (2) **2D detector**, (3) **Pose and State Estimator**. First, we briefly explain what is the objective and high-level functionality of each of the stages. Then, we go over each stage in more detail in the sections that follow. The explanation starts (and finishes) just before the Planning and Control stage. This choice is somewhat arbitrary since we can start the explanation almost anywhere in the feedback loop. Furthermore, it is assumed that the pushing has fully initialised. We further explain our implementation for initialising the pushing action in Section 4.

The **Planning and Control** stage is used to plan a trajectory for a finite time horizon and execute a control action for the current timestep. It takes the robot and object state vectors as an input. Additionally, it requires a desired goal location, which is set prior to initiating the pushing trial. This stage uses predictive modelling of kinematic robot dynamics and learned object dynamics to predict potential future paths based on sampled hypothetical control actions. It optimises the whole future trajectory based on a cost function and outputs an optimised control action. Then, the optimised control action is used to actuate the robot for the duration of 1 timestep. After this control action, the whole time horizon is shifted 1 timestep into the future and the planning, optimising and control is reiterated. This results in a continuous pushing motion. We extract a future object state prediction by re-using the optimised control action and the learned predictive object dynamics model.

The **2D detector** stage is used to calculate a Region of Interest (*RoI*). This is done by using the future object state prediction from the **Planning and Control** stage. This prediction is used to project the object in its predicted pose onto the image plane, from which a 2D bounding box is calculated. The bounding box used used to crop the image, resulting in the *RoI*.

The **Pose and State Estimator** stage is used to estimate the object's pose based on the *RoI*, using an off-the-shelf pose estimator and RANSAC/PnP. The is used again here for RANSAC/PnP initialisation. The pose estimator takes the *RoI* and finds a dense pixel-wise correspondences with the object's 3D model. These correspondences, along with the pose prediction from the **2D detector** stage, are processed by RANSAC/PnP which outputs a pose estimate. The resulting pose estimate is used to estimate the object's state, which is required by the **Planning and Control** stage.

3.1 Planning and Control Stage

The Planning and Control stage takes the state of the robot and object and outputs an optimised control signal $\mathbf{u}_{opt}$, which is sent to the robot actuators. For our implementation, we use a Model Predictive Path Integral (*MPPI*) controller taken from the work presented in [34]. This implementation of the MPPI controller has the following crucial attributes which we wish to exploit for our purpose:

- **Future state predictions**. The predictions that are used for control optimising can be exploited to accelerate the 2D detector, as we will explain in Section 3.2.
- **Same underlying data**. The predictive object dynamics model from [34] is trained on the same raw data that was used to train our pose estimator, which eliminates the requirement to train the model on new data.

We briefly introduce the essentials of *MPPI* control here, however we regard additional details on its functioning outside the scope of this work. For these details we refer to the original implementation [34].

MPPI control — MPPI is a control algorithm that combines elements of Model Predictive Control (*MPC*) and stochastic optimisation. MPPI samples a set of control action sequences based on the previous optimised control sequence. Each control action

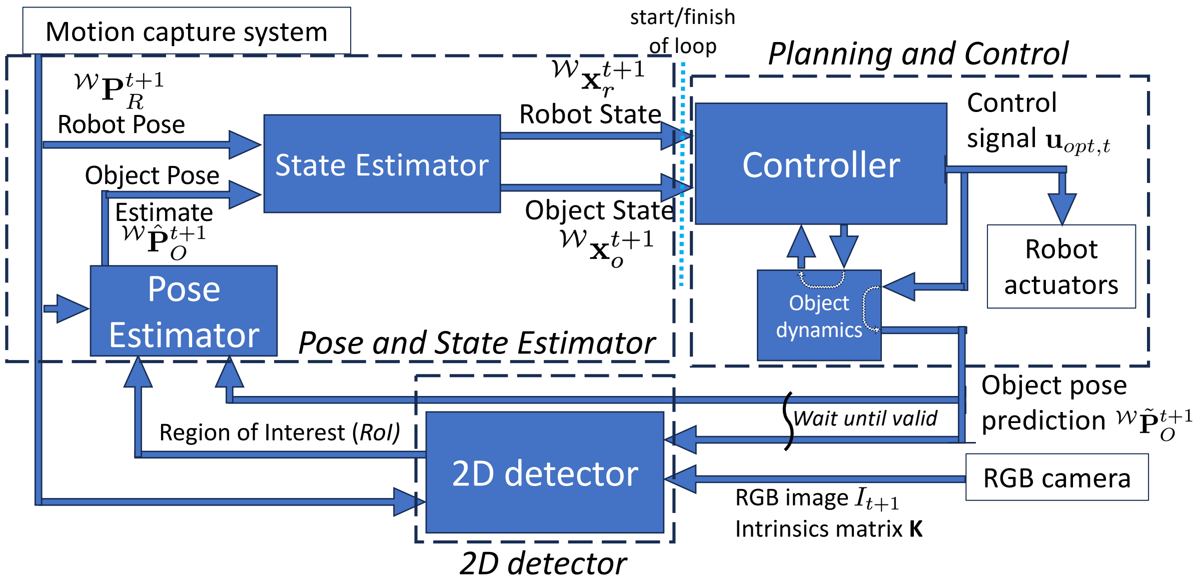

Motion capture system
start/finish of loop
Planning and Control
$^{\mathcal{W}}\mathbf{P}_R^{t+1}$ Robot Pose
$^{\mathcal{W}}\mathbf{x}_r^{t+1}$ Robot State
State Estimator
Controller
Control signal $\mathbf{u}_{opt,t}$
Object Pose Estimate $^{\mathcal{W}}\hat{\mathbf{P}}_O^{t+1}$
Object State $^{\mathcal{W}}\mathbf{x}_o^{t+1}$
Robot actuators
Pose Estimator
Object dynamics
Pose and State Estimator
Object pose prediction $^{\mathcal{W}}\tilde{\mathbf{P}}_O^{t+1}$
Wait until valid
Region of Interest (RoI)
2D detector
RGB camera
RGB image I_{t+1}
Intrinsics matrix K
2D detector

Fig. 1. Visualisation of our framework, as explained in Section 3.

within the previous control sequence is perturbed with noise from a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma})$. This perturbation is performed multiple times on the original optimised sequence, resulting in a set of U sampled hypothetical control sequences. Each control action consists of two terms: $\mathbf{u}_u^t = [v_u{}^t, w_u^t]^T \in \mathbb{R}^2$ for time-step t within the horizon H and sequence u within the set of U sequences where v is the forward velocity and w is the yaw velocity. $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are used to set the sampling distribution for each control input separately, so that $\boldsymbol{\mu} = [\mu_v, \mu_w]^T \in \mathbb{R}^2$ and $\boldsymbol{\sigma} = [\sigma_v, \sigma_w]^T \in \mathbb{R}^2$. Each control sequence has a length of H time-steps, which represents the finite time horizon for which the algorithm predicts. MPPI uses predictive models of the robot and object dynamics to simulate the propagating system behaviour. The robot model is based on a non-holonomic kinematics model and the object model consists of a neural network that has learned the object dynamics from real-world pushing data. For each sampled control action a cost is calculated, whose value is a weighted sum that represents various aspects of system behaviour, like control input magnitude and proximity to the desired goal location $\mathcal{G}(x, y)$. By weighting the actions based on their associated costs and minimising it, an updated optimised control sequence is formulated. Now, $\mathbf{u}_{opt}^{t=1}$ is used to actuate the robot until the next time-step's optimisation is performed. The remainder of the optimised control actions are used for the next iteration of the algorithm.

Obtaining the predicted object state — Because of the nature of the MPPI's optimisation process, the optimised trajectory is obfuscated and only the optimised control action $\mathbf{u}_{opt}^t$ for current time-step t is available. Therefore, we take $\mathbf{u}_{opt}^t$ and re-feed it into the predictive object dynamics model f_θ. This model uses Probabilistic Neural Networks (*PNNs*) [7], which is trained to predict the object's state $\tilde{\mathbf{x}}_o^{t+1} = [\tilde{x}_o, \tilde{y}_o, \tilde{\theta}_o, \tilde{v}_{xo}, \tilde{v}_{yo}, \tilde{\omega}_o]^T \in \mathbb{R}^6$ based on the current state vector $\hat{\mathbf{s}}^t = [\hat{x}_r, \hat{y}_r, \hat{\theta}_r, \hat{v}_r, \hat{\omega}_r, \hat{x}_o, \hat{y}_o, \hat{\theta}_o, \hat{v}_{xo}, \hat{v}_{yo}, \hat{\omega}_o]^T \in \mathbb{R}^{11}$ and a control action $\mathbf{u}^t = [v^t, w^t]^T \in \mathbb{R}^2$. Here, $[\hat{x}_r, \hat{y}_r, \hat{\theta}_r]$ are the robot's position and orientation in the world. $[\hat{x}_o, \hat{y}_o, \hat{\theta}_o]$ are the object's position and orientation in the world. $[\hat{v}_r, \hat{\omega}_r]$ are the robot's velocity and yaw velocity in the robot's frame. $[\hat{v}_{xo}, \hat{v}_{yo}, \hat{\omega}_o]$ are the object's translational velocity and yaw velocity in the world.

To use the object's predictive model for our method, we modify the *PNN* input pre-processing and output post-processing steps to process a single input control action $\mathbf{u}_{opt}^t$, as opposed to a multitude of hypothetical control actions. We can describe the second pass with

$$\tilde{\mathbf{x}}_o^{t+1} = (f_\theta(\hat{\mathbf{s}}^t, \mathbf{u}_{opt}^t)) \tag{1}$$

where $\tilde{\mathbf{x}}_o^{t+1}$ represents the predicted object state given the current system state $\hat{\mathbf{s}}^t$ and optimised control action $\mathbf{u}_{opt}^t$ from the MPPI optimisation. $f_\theta(\cdot)$ denotes the probabilistic neural network with θ the accompanying sets of trained weights. Finally, we take the pose parameters from the resulting state vector $\tilde{\mathbf{x}}_o^{t+1}$ to formulate the pose prediction $^{\mathcal{W}}\tilde{\mathbf{P}}_O^t$. Additionally, we add the previously estimated object pose's z component $\hat{z}_o^t$ to lift the representation to from 2D to 3D translation. The resulting predicted object pose can be described as

$$^{\mathcal{W}}\tilde{\mathbf{P}}_o^{t+1} = \begin{bmatrix} \cos\tilde{\omega}_o^{t+1} & -\sin\tilde{\omega}_o^{t+1} & 0 & \tilde{x}_o^{t+1} \\ \sin\tilde{\omega}_o^{t+1} & \cos\tilde{\omega}_o^{t+1} & 0 & \tilde{y}_o^{t+1} \\ 0 & 0 & 1 & \hat{z}_o^t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where the 4x4 homogeneous transformation matrix representation is used to be able to efficiently transform this pose prediction to different coordinate frames. This is crucial to our method, as we will explain in the following section.

3.2 2D detector stage

The objective of the 2D detector is to obtain a Region of Interest (*RoI*), which is required by the Pose Estimator stage. The *RoI* is a zoomed-in section of the image ideally containing the un-occluded part of object. By using our implementation, we achieve a 50x speedup over the FCOS 2D detector [41] used in the original ZebraPose [39] pose estimator, and we present these results in Section 4. We will now explain the details of our implementation over the course of 5 consecutive steps, which are visualised in Figure 2.

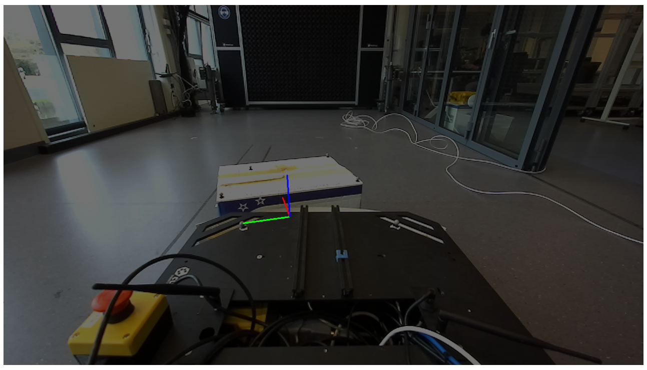
The pose prediction ${}^{\mathcal{W}}\tilde{\mathbf{P}}_{O}^{t+1}$obtained from the **Planning and Control** stage is valid only at time *t+1*. Therefore the system awaits the corresponding timestamp to arrive before performing the transformations between coordinate frames. Once the correct timestamp has arrived, the most recent robot pose ${}^{\mathcal{W}}\mathbf{P}_{R}^{t+1}$ is obtained. Then, the transformation to the camera frame $\mathcal{C}$ is calculated using the static transformation ${}^{\mathcal{R}}\mathbf{T}_{C}$ from the robot frame $\mathcal{R}$ to the camera frame $\mathcal{C}$. This is performed using canonical coordinate frame transformation using 4x4 transformation matrices and can be described by

$$ {}^{\mathcal{C}}\tilde{\mathbf{P}}_{O}^{t+1} = {}^{\mathcal{R}}\mathbf{T}_{C}{}^{-1}({}^{\mathcal{W}}\mathbf{P}_{R}^{t+1})^{-1}({}^{\mathcal{W}}\tilde{\mathbf{P}}_{\mathcal{O}}^{t+1}) \tag{3} $$

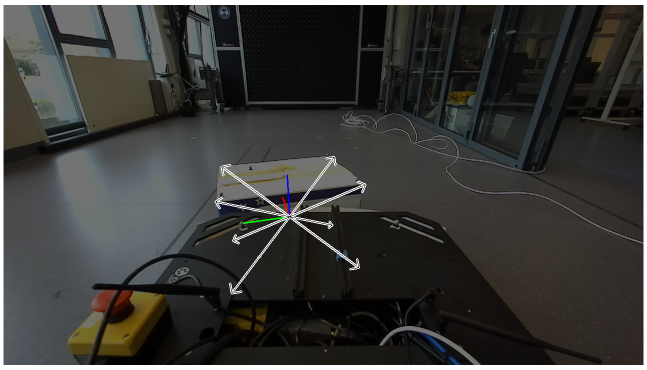
and this is visualised in Figure 2a.

We can describe the object's 3D bounding box in the camera frame $\mathcal{C}$ with $\tilde{\mathcal{B}}_{\text{3D}} = [\tilde{x}, \tilde{y}, \tilde{z}, \tilde{\theta}_x, \tilde{\theta}_y, \tilde{\theta}_z, w, l, h] \in \mathbb{R}^9$. The pose parameters $(\tilde{x}, \tilde{y}, \tilde{z}, \tilde{\theta}_x, \tilde{\theta}_y, \tilde{\theta}_z)$ can be derived from the predicted object pose ${}^{\mathcal{C}}\tilde{\mathbf{P}}_{O}^{t+1}$. Its dimensions (w, h, l) are obtained from manual measurements of the object. We emphasise that even though our object is geometrically equal to its 3D bounding box, this parameterisation is general and can be used for any rigid 3D object with arbitrary shape and topology.

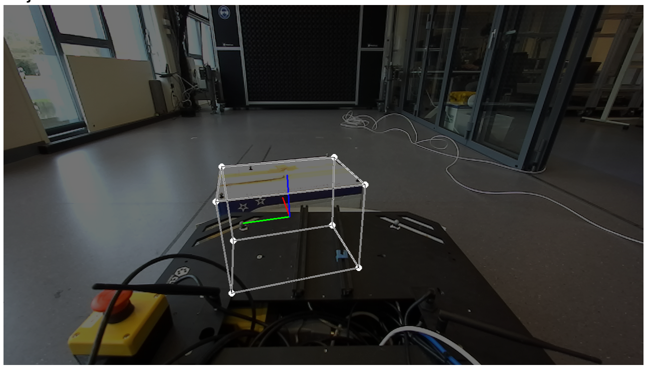
Using the 3D bounding box dimensions, we construct the 3D translations ${}^{\mathcal{O}}\mathbf{p}_i$ from the object's centre to each 3D bounding box corner, one translation for each corner $i = 1,..,8$. We construct the translation vector $[{}^{\mathcal{O}}x_i, {}^{\mathcal{O}}y_i, {}^{\mathcal{O}}z_i, 1]^T$ for the i-th corner by populating it with $\frac{1}{2}[\pm w, \pm l, \pm h, 1]^T$, for a total of $2^3 = 8$ possible variants. The 3D translation vector is described with a 4x1 vector to support the calculations using 4x4 transformation matrices. Now, the object-centric 3D coordinates are transformed to the camera frame $\mathcal{C}$ by coordinate transformations, which can be described by



(a) Visualisation of the predicted object pose ${}^{\mathcal{C}}\tilde{\mathbf{P}}_{O}^{t+1}$from **Planning and Control**, which is achieved by transforming the object pose prediction ${}^{\mathcal{W}}\tilde{\mathbf{P}}_{O}^{t+1}$ from the Planning and Control stage is from world frame $\mathcal{W}$ to camera frame $\mathcal{C}$.



(b) We use the 3D bounding box $\tilde{\mathcal{B}}_{\text{3D}}$ dimensions to calculate the static transforms between object centre and its 8 corners. The static transforms are shown as arrows. Then, they are transformed from object frame $\mathcal{O}$ to camera frame $\mathcal{C}$.



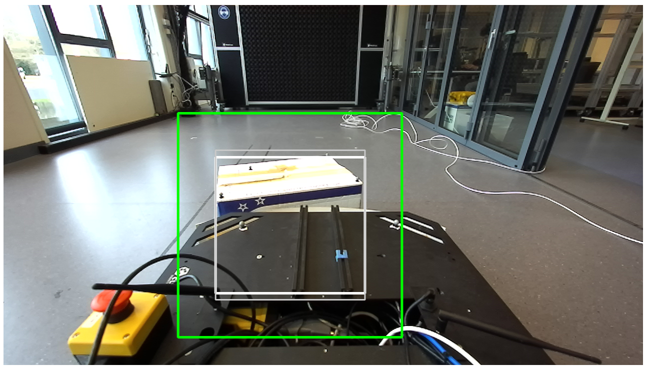
(c) By using perspective projection we can project the 3D bounding box $\tilde{\mathcal{B}}_{\text{3D}}$ corners onto the image plane. The predicted corner locations are shown as white dots. For visualisation purposes, we draw lines between the adjacent $\tilde{\mathcal{B}}_{\text{3D}}$ corners to clearly see the contours of $\tilde{\mathcal{B}}_{\text{3D}}$.

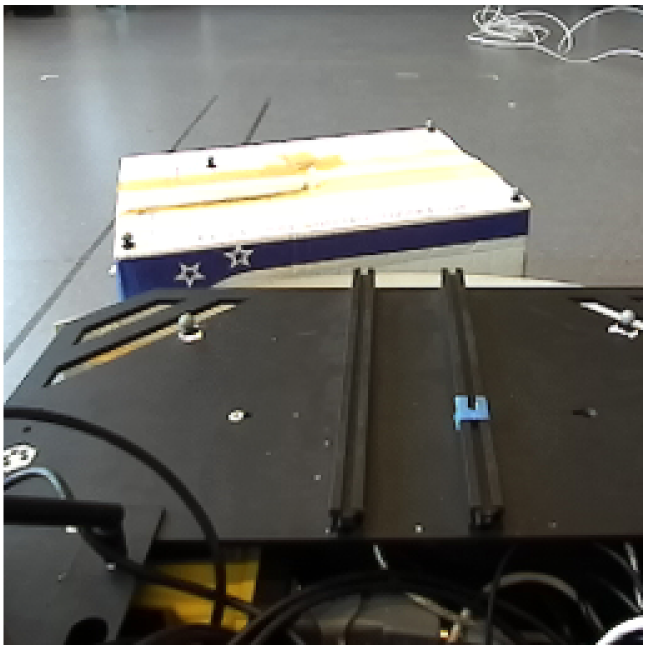
Fig. 2. Visualisation of the processing steps used in our 2D detector implementation. Please note that the RGB frames in these visualisations are only used for visualisation purposes and that actual pixel processing only occurs during the processing step from 2d to 2e.

$$ {}^{\mathcal{C}}\tilde{\mathbf{p}}_{i}^{t+1} = {}^{\mathcal{C}}\tilde{\mathbf{P}}_{\mathcal{O},t+1} \; {}^{\mathcal{O}}\mathbf{p}_i \tag{4} $$

$$ \begin{bmatrix} {}^{\mathcal{C}}\tilde{x}_i^{t+1} \\ {}^{\mathcal{C}}\tilde{y}_i^{t+1} \\ {}^{\mathcal{C}}\tilde{z}_i^{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{C}}\tilde{\mathbf{R}}_{\mathcal{O},t+1} & {}^{\mathcal{C}}\tilde{\mathbf{t}}_{\mathcal{O},t+1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} {}^{\mathcal{O}}x_i \\ {}^{\mathcal{O}}y_i \\ {}^{\mathcal{O}}z_i \\ 1 \end{bmatrix} \tag{5} $$

The resulting vectors ${}^{\mathcal{C}}\tilde{\mathbf{p}}_{i}^{t+1}$ represent the corners of


(d) By finding the maxima and minima of the $[\tilde{u}_i^{t+1}, \tilde{v}_i^{t+1}]$ pixel coordinates, we can calculate the coordinates of the perimeter of the 2D bounding box $\tilde{\mathcal{B}}_{2D}$, visualised with the white border. Then we square $\tilde{\mathcal{B}}_{2D}$(shown as gray border) and perform a padding operation, resulting in the Region of Interest *RoI* coordinates (shown as green border). We take the RGB image I_{t+1} and crop the green area.


(e) Finally, the *RoI* is resize to the desired size (256x256 RGB) using bilinear interpolation.

the 3D bounding box in the camera frame $\mathcal{C}$ and this is visualised in Figure 2b.

Now, we can project the 3D bounding box corner transformations from the camera frame $\mathcal{C}$ onto the image plane using perspective projection, which is used for mapping a point in 3D space onto an camera's image. For each corner i we perform the perspective projection as

$$\begin{bmatrix} \tilde{u}_i^{t+1} \\ \tilde{v}_i^{t+1} \\ 1 \end{bmatrix} = \mathbf{K}\ {}^{\mathcal{C}}\tilde{\mathbf{p}}_{i^{t+1}} \tag{6}$$

$$\begin{bmatrix} \tilde{u}_i^{t+1} \\ \tilde{v}_i^{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{\mathcal{C}}\tilde{x}_i^{t+1} \\ {}^{\mathcal{C}}\tilde{y}_i^{t+1} \\ {}^{\mathcal{C}}\tilde{z}_i^{t+1} \end{bmatrix} \tag{7}$$

where $\mathbf{K}$ is the camera intrinsic matrix, which is known for our camera. It consist of the focal lengths f_x and f_y, as well as the principal point offset c_x and c_y. We show the resulting pixel coordinates for each corner in Figure 2c. We connect adjacent corners to visually emphasise the 3D bounding box contours.

To obtain the 2D bounding box $\tilde{\mathcal{B}}_{2D}$, we define the upper-left corner as $[\min(\tilde{u}_i^{t+1}), \min(\tilde{v}_i^{t+1})]$ and the lower-right corner as $[\max(\tilde{u}_i^{t+1}), \max(\tilde{v}_i^{t+1})]$ and we draw the resulting 2D bounding box $\tilde{\mathcal{B}}_{2D}$ in white in Figure 2d. Then, the 2D bounding box is squared (increased in size according to its largest dimension), visualised in gray in Figure 2d. Next, a padding step is performed using a padding ratio r_{pad}, resulting in the *RoI* pixel coordinates, shown in green in Figure 2d.

Finally, the RGB frame I_{t+1} is cropped to the resulting range of selected pixels and resized to nxn pixels using bilinear interpolation [22]. This is shown in Figure 2e. n is the dimensions corresponding to the input size of the subsequent pose estimator, explained in the following section.

3.3 Pose and State Estimator Stage

The pose estimator stage consists of two main steps: 1) estimate the object's pose using a Region of Interest (*RoI*) image containing the target object. 2) The controller requires both the pose and the velocities of the system, so we use a state estimator to estimate the updated object velocities (with planar and yaw velocities), using the previous and current pose estimate. For estimating the pose, we select ZebraPose [39], which is a state-of-the-art learning-based object pose estimation method. We select this specific method for the following reasons: 1) Since our robot is equipped with an RGB camera sensor, we require a method that is developed specifically for RGB input, as opposed to the more widely researched RGB-D format. 2) It has state-of-the-art performance within RGB methods. 3) The source code of the implementation was available on GitHub. We will provide a high-level explanation on its functionality. For any further details, we refer to the original work [39]. We use the state estimator implementation from [53], following [34]. It uses a Kalman Filter to perform the state estimation. We explain the state estimator as part of this stage to improve readability, even though it is only used for the functioning of the **Planning and Control** stage.

ZebraPose estimator — The *RoI* from the **2D detector** is normalised (Figure 3a), and fed through a DeepLabV3 [6] semantic segmentation Convolutional Neural Network (*CNN*), which was trained to output a stack of 17 binary masks. The first mask is used for estimating the presence of the entire object inside the image for each pixel, as is shown in Figure 3b. The remaining 16 masks represent a course-to-fine segmentation scheme, which is shown in Figure 3c. Then each pixel is analysed canonically, from the first (course) layer in the stack to the last (fine), resulting in a 16-bit binary sequence containing the pixel's values for each binary layer. The network is trained to output the correct binary sequence for each pixel within the object mask. Each binary sequence corresponds is decoded into its decimal counterpart, which corresponds to a point on the surface of the ground-truth 3D model. This 3D model is visualised by converting the 16-bit binary sequence into a red-green encoding, as is shown in Figure 4b. Essentially, the network is predicting each pixel's class with a total number of $2^{16} = 65.536$ possible classes. Since the 3D model is also segmented into 65.536 regions, a pixel match can be found for each surface patch. To realise the match quickly, a *class correspondence dictionary* is created. This dictionary contains a list of class IDs (decimal number corresponding to the base-2 binary sequence) and corresponding 3D coordinates. The pixel's estimated binary code is decoded to the decimal number, which is looked up in the *class correspondence dictionary*, matching it to a 3D object surface coordinates. All matched pixel correspondences are aggregated into a list of 2D-3D correspondences.

RANSAC/PnP — The list of 2D-3D correspondences is processed by RANSAC/PnP, which will perform both outlier filtering (RANSAC [12]) and pose solving using an iterative PnP algorithm that uses a Levenberg-Marquardt (LM) minimisation scheme [28]. Additionally, we supply the algorithm with an initial pose guess. This step can help initialising the algorithm, which increases the accuracy obtained per iteration, as we will show in Section 4. As the initial guess, we use the pose prediction ${}^{\mathcal{C}}\tilde{\mathbf{P}}_O^{t+1}$, obtained from the 2D detector stage. After j number of iterations, we obtain the resulting object pose estimate ${}^{\mathcal{C}}\hat{\mathbf{P}}_O^{t+1}$.

Finally, the newly acquired pose estimate ${}^{\mathcal{C}}\hat{\mathbf{P}}_O^{t+1}$is transformed to the world frame $\mathcal{W}$ by using the robot's pose ${}^{\mathcal{W}}\mathbf{P}_R^{t+1}$ doing

$$ {}^{\mathcal{W}}\hat{\mathbf{P}}_O^{t+1} = {}^{\mathcal{W}}\hat{\mathbf{P}}_R^{t+1} \; {}^{\mathcal{R}}\mathbf{T}_C \; {}^{\mathcal{C}}\hat{\mathbf{P}}_O^{t+1} \tag{8} $$

State Estimator — Following [34], we use the State Estimator implementation from [53] to obtain the updated object and robot state vectors $[{}^{\mathcal{W}}\mathbf{x}_o^{t+1}, {}^{\mathcal{W}}\mathbf{x}_r^{t+1}]$. It uses an Unscented Kalman Filter (UKF) to estimate the translational and angular velocities, based on the previous and current pose estimates. The state vectors are sent to the Planning and Control stage, which concludes the description of the feedback loop.

4 EXPERIMENTS

We conduct a series of real-world experiments using a mobile base equipped with a calibrated on-board camera. The data which was collected during the experiment serves as a dataset for further experiments and analysis. First, the effectiveness of using our time-coherent 2D detector approach is demonstrated by comparing it to a baseline method. Then, the co-dependence between the pushing controller and pose estimator is explored. We conclude our experiments by assessing the real-world pushing performance of the unified framework, compared to the reproduced baseline pushing method.

	Minimum value	Maximum value
θ_r	-5°	5°
θ_o	-30°	30°
${}^{o}y_r$	-$\frac{1}{2}L_{box}$	$\frac{1}{2}L_{box}$

TABLE 1. Minimum and maximum values for randomised pushing initialisation during real-world pushing experiments.

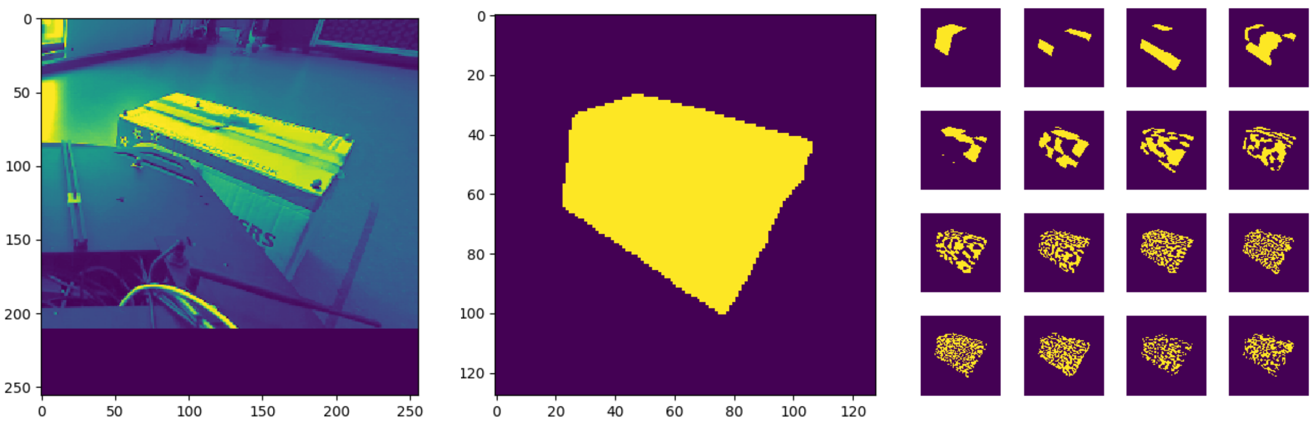
4.1 Experimental setup

The process of experimental data acquisition and subsequent post-processing is described here. We post-process the data to re-use it for further experiments, which has the objective to enable fair comparison between different system configurations.

Real-world experiments — Real-time pushing experiments are performed in a MoCap lab to measure the ground-truth pose data of the robot and object, so that the performance of our proposed method can be evaluated. Meanwhile, the raw data is recorded for further experiments and comparison to baselines and between different framework configurations. In total, we record 70 trials in which the robot is given the task to push the object to a desired goal location. We measure the ground truth using reflective markers and the room's MoCap system. 35 trials are used to set a baseline pushing performance using the work presented in [34]. During the remaining 35 trials, our proposed framework is deployed using the parameters presented in Appendix A. All data is recorded to ROS bagfiles.

We create the different initial conditions by manually manipulating the initial object θ_o and robot angles θ_r and the y-position of the robot, relative to the object ${}^{o}y_r$ at the start of each trial. The x-position of the robot is chosen such that the robot and object are in contact. We vary the initial conditions of the box randomly within the parameter boundaries presented in Table 1. The pose estimator's RANSAC/PnP iteration parameter is set to 150 and we omit the initial pose guess. The parameters used for the pushing controller are not changed between the two experiments and can be found in Appendix A. We select two target goal locations, which are a straight push and left push: $[5.0, 0.0]$ and $[4.0, 2.0]$, which are visualised in Figure 5.

Filtered recordings — The real-world experiment bagfiles are filtered to exclude all inference information (predictions and estimations), which allows additional real-time experiments to be conducted on the same data as the real-world experiments. This is achieved by playing back the filtered bagfile and running the framework on the data during real-time playback. We again record these experiments and



(a) Example *RoI* from the **2D detector** stage image after pre-processing for CNN input **(b)** Binary mask of entire object, as estimated by the CNN **(c)** All 16 binary masks which represent a course-to-fine semantic segmentation

Fig. 3. The input and outputs of the pose estimator's *CNN*.

save them to ROS bagfiles, similar to the real-world experiments.

Dataset format — The recorded bagfiles from the real-world experiments are post-processed by converting them into the BOP dataset format using our BOP-ROS conversion toolkit [43]. This allows quantitative analysis using our adaptation [44] of the BOP toolkit, permitting us to evaluate the pushing controller and pose estimator in a unified approach.

By using this data format we can generate ground-truth bounding boxes, akin to the *AIRlab* dataset (Section 4.2.2) used for the training of the pose estimator. This dataset is subsequently used for offline inference using the test script from the modified ZebraPose repository [46]. Then the resulting pose estimates are assessed using the BOP toolkit [44] yielding our baseline pose estimator performance measurements. Since our method integrates predictions based on a predictive model which requires velocities, we cannot test it on currently existing 6D pose benchmarks. We argue that since our training strategy is similar to the original ZebraPose [39] method, the baseline pose estimator performance is a representative reflection of the original method and can be used as a baseline for comparison.

4.2 Implementation details

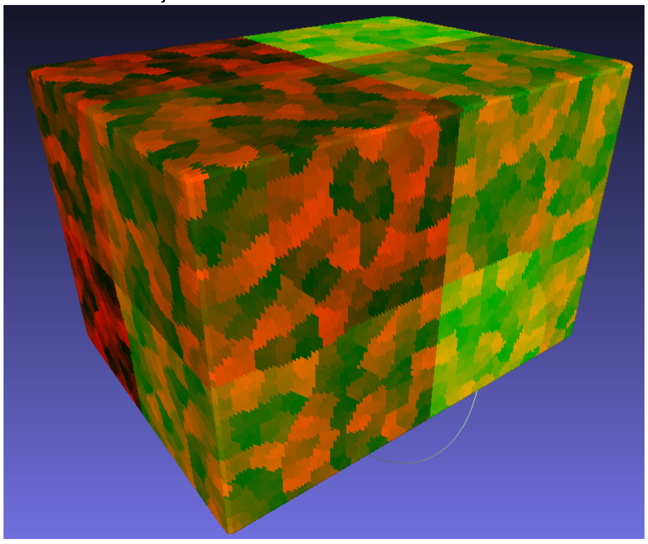
4.2.1 *Hardware and software*

The experiments were performed in a MoCap lab using a Clearpath Robotics Husky A200 robot and a conventional moving box (490 x 330 x 365mm) with a weight of 4kg, equally distributed. The framework is deployed using our ROS architecture (adapted from [34]) executed from a notebook running Ubuntu 20.04, equipped with an Intel 6-core 2.60 GHz CPU and Nvidia Quadro P2000 4GB GPU with CUDA 11.6. In Appendix B we present the framework overview from a ROS software perspective. The MoCap system consists of 16 infrared cameras (OptiTrack Flex 17W) that supply the real-time information regarding the 6 DoF poses of the robot and the object. The publishing of the real-time pose data is done using the OptiTrack's Motive software. The system runs at a frequency of 120 Hz. We track the position of the robot and object using 6 reflective OptiTrack markers for each rigid body (Robot and object). We perform a full OptiTrack calibration process prior to the execution of the experiments.

4.2.2 *AIRlab dataset*

For training the pose estimator network, we record our own dataset in the *AIRlab* in Delft. During this recording, the control data and dynamics of the object are recorded, which were used for [34]. The data is time-synchronised and post-processed. Training instances with a small robot movement compared to the previous timestep are discarded to prevent duplicate training images. Following [14], the data is divided into a 80/20 train/test split, while equally distributing the images of left turns, right turns and straight pushing manoeuvres. This resulted in our *AIRlab* dataset consists of 4264 training and 1066 testing images. Then, the data is converted to BOP format [14] using our BOP-ROS conversion toolkit [43]. For this conversion we create a pseudo-depth image using the ZED stereo camera and the ZED SDK software. We use the built-in depth output of the SDK and take a single high-quality depth image. This depth image contains the contours of the robot's bumper, which the BOP toolkit uses to calculate the part of the object that is visible or occluded. The depth image is created only once and used to generate all the masks. Then the entire object mask and visible mask can be generated. These are required for training the ZebraPose pose estimator. We show an example training instance in the different data formats in Figure 6

(a) 3D model of the object, textured with high quality images from the real-world object.



(b) Textured 3D model of the object used for generating training data for the ZebraPose pose estimator. The object surface is divided into patches corresponding to a 16-bit binary sequence. This is visualised with an 8-bit/8-bit red/green colour encoding.

Fig. 4. Textured 3D models of the object used for generating training data for the ZebraPose pose estimator.

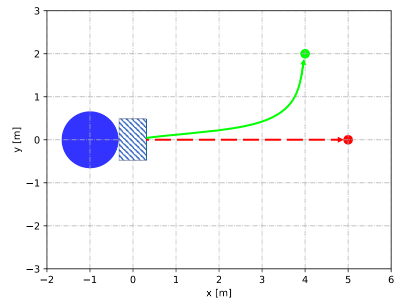


Fig. 5. The two possible goal coordinates from the real-world experiments are visualised with trajectories in red (straight) and green (left turn). The robot is visualised by the blue circle and the object is visualised by the hatched rectangle.

Parameter	Value
Epochs	4000
Batch size	8
ResNet layers	34
ResNet input size [pixels]	256x256x3
ResNet output size [pixels]	128x128x17
Predict Entire Object Mask	True
Learning rate	2e-4
Optimizer	Adam[21]
Data Augmentation	True[1]

TABLE 2. Training parameters for training the ZebraPose pose estimator on the *AIRlab* training split. [1]We present more details on the augmentation methods in Appendix D.

4.2.3 Network training strategy

ZebraPose [39] uses a modified DeepLabV3 semantic segmentation network [6] and we train it on our collected *AIRlab* dataset. We present the training parameters in Table 2, staying true to the original method [39]. To prevent overfitting, we reduce the number of epochs to 4000 and empirically find the highest performance when using a batch size of 8. We train the pose estimator with a ResNet-34 backbone using the ground-truth bounding boxes, which are subject to translational noise augmentation using a Gaussian distribution, which reduces the sensitivity to noisy bounding box estimates. During training, the input image is loaded and cropped using the noise-augmented and padded bounding box coordinates. Then, it is squared and finally resized to the desired 256x256 pixels using bilinear interpolation. This process is equal to the step describing the *RoI* calculations in 3.2.

Following [14], training instances in which the object is less than 20% visible are skipped. This percentage is based on the fraction of visible (un-occluded) object pixels, calculated from the masks (shown in Figure 6d and 6e) generated during BOP dataset creation.

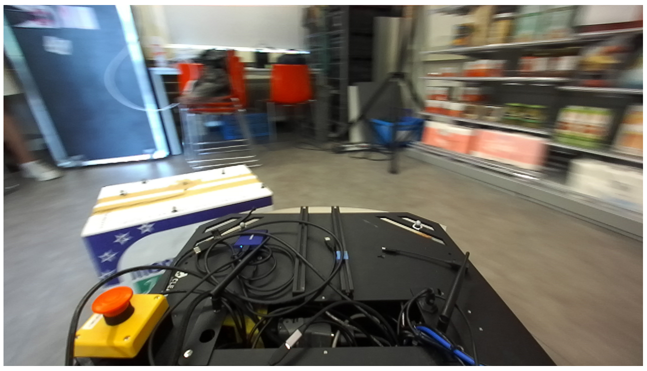
We perform several types of data augmentation, which we list in detail in Appendix D. Training was performed on a DelftBlue HPC [10] GPU node using 1x Nvidia Tesla V100 32GB GPU, and took 2.5 hours to train on our box object.

4.3 Performance Criteria

Pose estimator and *PNN*-prediction accuracy — To quantitatively compare the pose estimator and *PNN* predictions, we use the ADD (Average Distance of Points) [13]. This metric represents the average Euclidean distance between the 3D object model points in the estimated pose and their corresponding points in the ground-truth pose. It is defined as

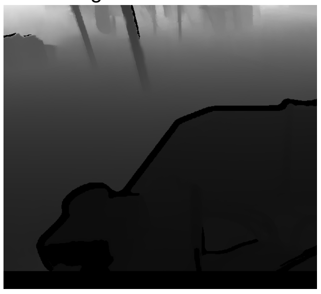
$$\mathrm{ADD} = \frac{1}{n}\sum_{i=1}^{n} ||(\mathbf{R}x_i + \mathbf{T}) - (\tilde{\mathbf{R}}x_i + \tilde{\mathbf{T}})|| \qquad (9)$$

where n is the object's total 3D model vertices, x_i is the i-th vertice of the 3D model. $[\mathbf{RT}]$ and $[\tilde{\mathbf{R}}\tilde{\mathbf{T}}]$ are

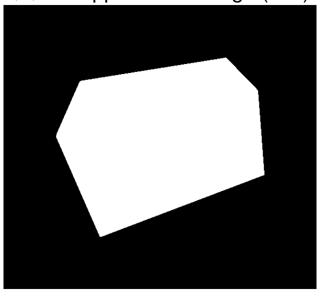


(a) Original RGB image

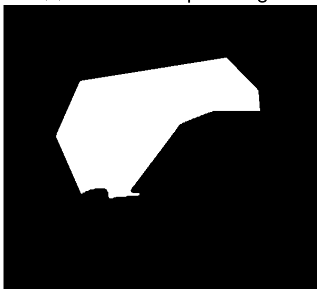
(b) Cropped RGB image (*RoI*)



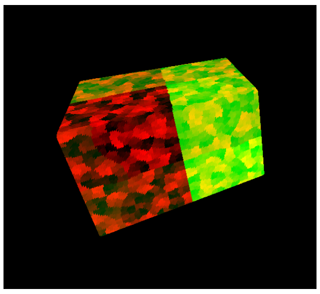
(c) Pseudo-depth image



(d) Entire object silhouette mask



(e) Visible object silhouette mask



(f) GT image

Fig. 6. Training instance from the *AIRlab* dataset. Figures 6b to 6f are cropped according to the processing steps from [39], which resembles the *RoI* calculation for training. Figure 6f shows a render of the textured 3D model (Figure 4b) in its ground-truth pose.

the respective ground-truth and estimated/predicted transformation matrices.

We additionally calculate the translational and rotational errors of the *PNN* predictions (matching it to the ground-truth pose for its valid timestamp), following [34]. These can be used to compare with the original work [34]. We do note that contrary to our analysis, the work of [34] does not take the z-dimension of the world into account.

AUC of ADD 0.1d — The majority of recent object pose estimation literature adopts variants of *ADD* to normalise its value according to the object's dimensions, which allows fair comparison between object classes with different object dimensions. To allow comparison to other works, we include the *AUC of ADD 0.1d* in our analysis of the pose estimator. We do not include the *Average Recall of ADD 0.1d*, since its value is 1.0 or extremely close to 1.0 for all our tested system configurations, thus its added value to our assessment is limited.

To calculate *AUC of ADD 0.1d*, the *ADD* for each pose estimate is calculated and classified it as either a correct estimation or an incorrect estimation based on a maximum allowable distance threshold. The average recall (AR) is the percentage of correct estimates for the chosen threshold. By varying the maximum allowable distance threshold and calculating the *average recall* at each threshold, we can plot a curve, containing the *average recall* value corresponding to a threshold ranging from 1% up to 10% of the object's diameter *d*. The *AUC* is then calculated by measuring the area under this curve. The closer the *AUC* value is to 1, the better the algorithm performs in terms of accurately estimating the object's pose. The diameter *d* is defined as the two most distant 3D points of the object's CAD model.

2D detector — Since we attempt to replace the 2D detector, we wish to assess the quality of the bounding box when using our method, compared to the pre-calculated ground-truth Bounding Box. We select the Intersection over Union (IoU), which is defined as

$$IoU = \frac{|\mathcal{B}_{est} \cap \mathcal{B}_{gt}|}{|\mathcal{B}_{est} \cup \mathcal{B}_{gt}|} \tag{10}$$

where $\mathcal{B}_{est}$ and $\mathcal{B}_{gt}$ are the respective estimated and ground-truth bounding box. $\mathcal{B}_{est} \cap \mathcal{B}_{gt}$ and $\mathcal{B}_{est} \cup \mathcal{B}_{gt}$ represent the intersection and union of the bounding boxes.

The *IoU* metric presents an intuitive value for the quality of the bounding box estimate between 0 (no overlap at all) and 1 (100% match). We take the mean of all calculated *IoUs* for each configuration, which is denoted as *mIoU*.

Pushing performance — Following [34], we present the following metrics for evaluating the performance of the pushing controller for MoCap-based pushing and pose estimator based pushing:

- Success rate
- Accuracy
- Time

We assess the success of pushing based on the final position of the object, and require it to be below a predetermined threshold of 1.0 m. Furthermore, we verify that the robot maintained contact with the object throughout the pushing and we require that the object was delivered within 30 seconds. If all these conditions are met, the pushing is considered successful. The accuracy is defined as the distance between the desired goal and the achieved final position in the horizontal plane. The time is defined as the duration between the first control action and the final control action.

4.4 Experiment 1: What are the effects of replacing the computer vision 2D detector with a prediction-based implementation on runtime and pose estimation performance?

We investigate the effect of our 2D detector on pose estimator performance, which we evaluate by means of 2D detector speed and resulting pose estimation accuracy. We set a baseline configuration by running inference on the ground-truth bounding boxes (*dataset format*) since we did not train the FCOS [41] detector on our dataset. We manually add 55ms to the inference time of the 2D detector in this configuration to imitate the runtime of computer vision based 2D detection using FCOS [41]. Then, we compare two variants of our implementation. In the first variant, the 2D bounding box is calculated using the real-time MoCap object pose as a source for the transformations and projection. This simulates the 'perfect' *PNN* prediction, since it is the most accurate pose of the object that can be achieved in this setup. The bounding box calculation is performed according to the steps presented in Section 3.2. The time that the system waits for the prediction to become valid is skipped, since the MoCap publishes the current object pose in real-time. For the second variant, the bounding box calculation is based on the *PNN* state prediction, which is our implementation as proposed in Section 3.2. All system configurations use 25 RANSAC/PnP iterations for pose estimation.

In Figure 7 we show a qualitative comparison of bounding boxes calculated from MoCap or *PNN* predictions. Additionally, the resulting pose estimate is rendered in the image.

In Table 3 a quantitative comparison is presented for the pose estimator performances. We notice that by using our implementation, a 50x-90x faster runtime can be achieved, depending on the configuration. The accuracy of the pose estimates does not suffer when qualitatively lower bounding boxes are being used.

4.5 Experiment 2: What are the effects of increasing pose estimator accuracy on subsequent prediction accuracy in object pushing when we compromise inference speed?

The co-dependency between the Planning and Control stage pose predictions and the Pose Estimator stage is studied. First, we vary the speed and accuracy of the Pose Estimation stage. By increasing the number of RANSAC/PnP iterations in the pose estimator, we can increase the accuracy while decreasing its speed. By comparing these co-dependent effects over a range of RANSAC/PnP iterations, we expect to gain understanding of the framework's potentials and current limits, while keeping the results realistic by reflecting real-world trade-offs. We conduct real-time experiments on the *filtered recordings* data. By running the framework on the same data-sequences for each configuration, the performance of the pose estimator and the subsequent predictions of the object's future state can be compared as equally as possible. We visualise the results in Figure 10.

Pose estimator — By increasing the number of RANSAC/PnP iterations, we achieve higher accuracy pose estimates at the cost of inference speed. The returns are diminishing after 25 iterations and disappear completely after 100 iterations.

***PNN* predictions** — We generally observe that a longer pose estimator inference has a negative effect on the accuracy of *PNN* predictions. On the other hand, there is a notable drop-off in prediction accuracy when the number of RANSAC/PnP iterations is below 25, when the accuracy of pose estimates begins to drop off.

4.6 Experiment 3: Can the pose prediction be used to increase the efficiency of RANSAC/PnP?

In this experiment the effects of initialising it with the pose prediction from the **Planning and Control** stage are investigated. Similar to Experiment 2, we run our framework using different quantities of RANSAC/PnP iterations, with and without initialising RANSAC/PnP with the valid prediction ${}^{C}\tilde{\mathbf{P}}_{O}^{t}$. Each configuration is ran on the same *filtered recordings* data sequences to fairly compare the two situations. Additionally, we repeat the experiment with a variant of our framework, where the **Planning and Control** stage is set to its baseline setting (based on MoCap object pose data). This results in pose predictions with a higher accuracy than when using the proposed feedback loop framework.

In Figure 9 the results are shown for using the pose predictions to initialise RANSAC/PnP when using our proposed framework. We include the *PNN* prediction accuracy as a reference to understand the quality of the prediction that is being supplied to the RANSAC/PnP algorithm. Our framework (Figure 9a shows no positive effect from using these predictions. Then, we show the results from initialising RANSAC/PnP using *baseline PNN* predictions, which are based on MoCap object pose data (Figure 9b). Now, an increase in pose estimate performance can be observed across the full spectrum of RANSAC/PnP iterations.

4.7 Experiment 4: What is the pushing performance of the proposed framework in a real-world scenario?

In this final experiment, the performance of pose estimation based object pushing is evaluated and compared to MoCap-based baseline pushing performance.

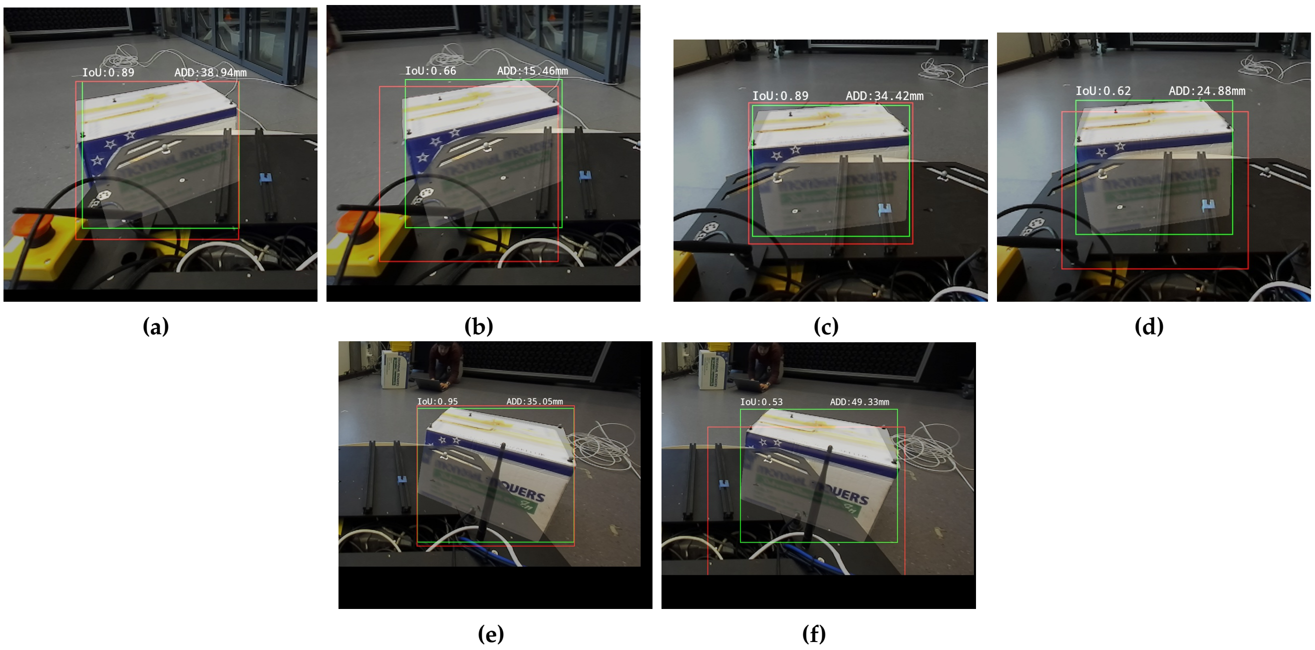


Fig. 7. Qualitative comparison of pose estimates from experiment 1. We compare the two configurations that use our proposed transformation and projection scheme. In Figures 7a, 7c and 7e, the bounding box calculation is based on MoCap. In Figures 7b, 7d and 7f, *PNN* predictions are used for bounding box calculation. The green border resembles the ground-truth bounding box and the red border is the bounding box resulting from our The *IoU* and ADD are shown in each image. We also render the object's 3D model in the estimated pose. As can be seen, no correlation can be observed between bounding box quality and pose estimation accuracy. We quantify this result in Table 3

Pose estimator	Bounding Box source	mIoU	ADD (mm)	AUC of ADD 0.1d	Total inference time (ms)	2D detector inference time (ms)
Baseline	Ground-truth	1.0000	22.34	0.8051	159	55.0*
Ours	OptiTrack	0.9202	22.15	0.8064	106	0.6
Ours	PNN	0.8246	21.87	0.8094	111	1.2

TABLE 3. Performance of the pose estimator when using different *RoI* acquisition methods. *mIoU* is the mean Intersection over Union for the bounding box, compared to the ground-truth (GT) bounding box. We present the *AUC of ADD 0.1d* and the mean ADD and compare it to inference time of the 2D detection. *We adopt the inference time of the Baseline configuration's 2D detector from the original method [39], which is based on the FCOS [41] detector. In reality, the FCOS 2D detector inference time is likely higher than 55ms on our system. We underline each metric's best score.

We conduct a series of 35 baseline experiments using MoCap to provide the object's pose information to the Planning and Control stage. Then, we perform 35 experiments using our proposed framework that incorporates object pose estimation. The objective is to assess how our framework compares to the baseline approach, and to quantify differences between MoCap-based pushing and pose estimation-based pushing. By comparing the results of these experiments, we gain insight into the effectiveness and reliability of our proposed framework in guiding the robot's actions based on estimated object pose. This investigation allows us to analyse the performance of the pose estimation approach and evaluate its suitability for real-world applications of object pushing.

In Table 4 we present the pushing performance of the two compared configurations. We observe a relative decrease of 8.9% in success rate, going from 31.4% to 28.6% when employing our method. The baseline method has a higher mean accuracy of 0.363m, as opposed to 0.510m when using our method, while the accuracy variance remains similar. The average time required to reach the end-goal is comparable. Furthermore, there is an increase in both translational (↑411%) and rotational (↑239%) error.

5 Discussion

We demonstrate the effectiveness of our proposed method to substitute the 2D detector from a state-of-the-art object pose estimator, making it more suitable for a real-time robot pushing scenario. It is demonstrated that a speedup of more than 50x can be achieved while the pose estimation accuracy does not suffer, which shows that using a 2D detector without keeping in mind temporal coherence or dynamics is an inefficient practice. Therefore, we argue that considering the time-dependent dynamics of robotic systems is beneficial in further robotic system development. As was shown in this work, a holistic approach to a problem may leads to improvement that would be more difficult to achieve when isolating challenges in robotics. Now, we will reflect on the results that are demonstrated in Section 4.

2D detector — Even though there is a degradation in 2D bounding box quality when using *PNN*-based

MPPI pushing controller configuration	Success rate (%)	Accuracy (m)	Time (s)	Translational error (mm)	Rotational error (deg)
Baseline (OT-based) [34]	31.4	0.363 ± 0.140	18.591 ± 7.402	9.8	2.36
Ours (PE-based)	28.6	0.510 ± 0.120	17.963 ± 6.200	40.31	5.64

TABLE 4. Real-world pushing results of our method compared to the reproduced baseline method [34]. OT=MoCap PE=pose estimator.

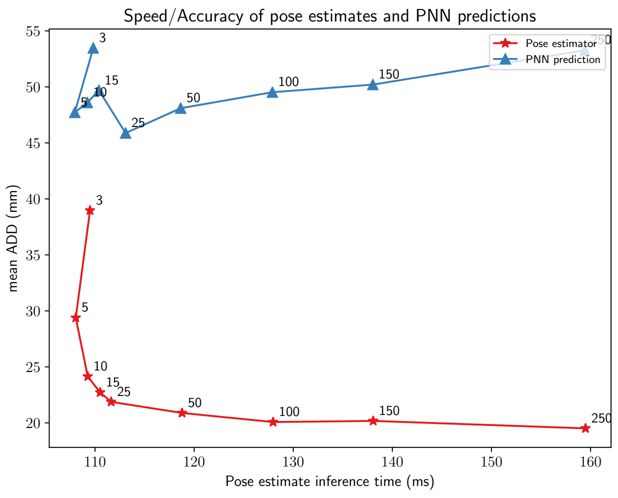
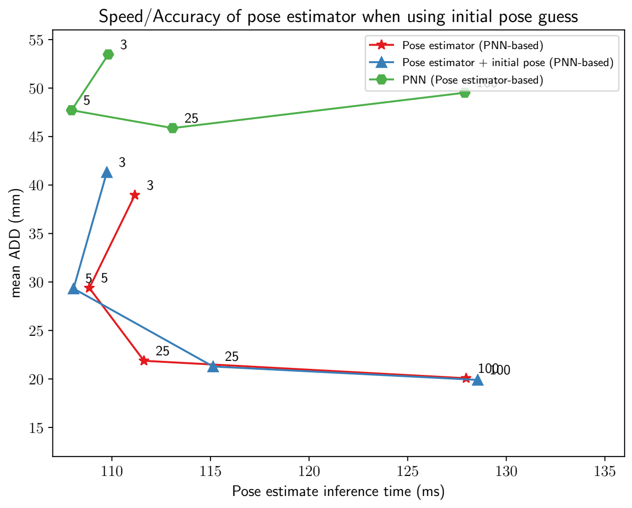
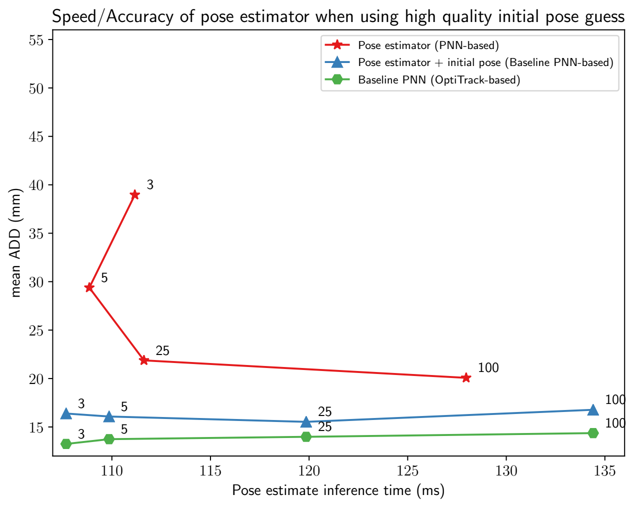


Fig. 8. The accuracy of the pose estimator and *PNN* predictions is assessed across a varying quantity of RANSAC/PnP iterations. The number at each marker represents the number of RANSAC/PnP iterations. Please note that the x-axis strictly represents the pose estimator inference time, thus we align the *PNN* x-position to its corresponding pose estimator inference time for visualisation purposes. We represent the accuracy for the estimates and predictions using the mean *ADD*.



(a)



(b)

Fig. 9. The effects of using the **Planning and Control** pose prediction to initialise the iterative RANSAC/PnP algorithm are visualised here. We show both the average accuracy of pose estimates and the average accuracy of the pose prediction that was used to initialise RANSAC/PnP. In 9a we use the pose prediction which is based on the previous pose estimate. No positive effect can be observed from using these pose prediction as an initial pose guess. In 9b we show a configuration in which pose predictions resulting from OptiTrack are used, where a beneficial effect can be observed.

bounding boxes, the resulting pose estimate does not suffer (Table 3) The reduction in runtime compared to the baseline configuration is due to the pose projection model's efficiency, compared to using a learning based computer vision 2D detection algorithm. A notable result is the inference time of the *PNN*-based configuration, which is higher than the OptiTrack-based configuration. Since it was evaluated during real-world pushing (as opposed to real-time inference on played back data), more data processing and coordinate transformations are performed in real-time by the computer. This results in a higher demand for computational resources during these experiments, which limits the speed of the pose estimator and 2D detector.

Runtime — The reported runtimes of the original ZebraPose components in [39] are significantly lower than our runtime analysis shows. For example, the authors report only 4ms for 150 RANSAC/PnP iterations, while our system requires 34ms on average for the same number of iterations. This reflects the CPU performance difference. In addition, our CNN + correspondence matching analysis shows 84ms, while the authors report 52ms for this step. This is related to GPU performance difference, since the CNN computations are performed on the GPU. We can argue that our implementation may be faster than 0.6ms on systems with more computational overhead and/or a faster CPU. Likewise, this means that the 2D detector is likely slower than 55ms using our system. Even though we argue that discussing potential speed improvements can be regarded as speculative, we want to highlight that our implementation has untapped potential. The delay in pose estimations significantly hampers the system's performance, and addressing it can yield further improvements.

Co-dependency of predictions and estimations —

By exploring the co-dependency of the framework's stages, it becomes clear that a balance between pose estimator accuracy and speed has to be found to optimise the performance of the system. While the accuracy of the pose prediction can never exceed that of the preceding pose estimate, it certainly is compromised by its tardiness. This further emphasises the importance of minimising computationally heavy operations in real-world robotics settings.

As we showed, both the tardiness and accuracy of a pose estimate under the circumstance of continuous pushing play important roles in the accuracy of the subsequent *PNN* pose predictions. While the 3-15 RANSAC/PnP iteration configurations all operate within a similar time window, we still see a drop in the accuracy of the resulting *PNN* predictions, which reflects the drop in pose estimator accuracy.

So far, this only shows a one-sided dependency, since the accuracy of the pose estimator has shown not to be influenced by lower quality bounding boxes resulting from inaccurate pose predictions. However, as the experiments regarding the RANSAC/PnP initial pose guess showed, using high quality pose predictions can influence subsequent pose estimates (Figure 9b). Still, this positive effect can only be achieved in similar implementations using RANSAC/PnP with low latency, where it will merely provide stability, since the accuracy of the pose prediction cannot exceed the accuracy of the preceding pose estimate.

Real-world pushing performance — When testing our framework in a real-world pushing scenario, we observed significant increases in the translational (↑411%) and rotational (↑239%) errors of the pose prediction, when compared to the baseline method (Table 4. This result correlates with an increase in end-goal error (↑40% relative). Furthermore, a less prominent decrease in success rate was measured (↓9% relative), and no degradation of the time required to finish the pushing task could be observed.

Foremost, this results demonstrate the robustness of the controller to deal with increased noise in the pushing predictions as a result of additional inaccuracies in the object pose information. Furthermore, we argue that even though the relative increase in errors are large, when the absolute resulting error is not, since a translational error of 41mm is only 6% of the object's diameter (687mm). We do believe that these inaccuracies may have a larger impact on the pushing performance when different objects are used, which have less desirable characteristics (centre of gravity, weight distribution). We might even observe a deterioration of pushing when the box would be rotated a quarter turn, since the pushing characteristics. Nevertheless, we successfully pushed an object to the end location for a number of trials that is comparable to the baseline, which we regards as an overall success or proof on concept of our work.

5.1 Motion Capture-less pushing

The knowledge of a robot pose is assumed to be available within the scope of this work. We implemented it using the MoCap. We acknowledge that implementing an alternative method for obtaining this information is essential to be pushing fully MoCap-less. This step inherently introduces additional errors and/or noise to the framework, which results in its own set of limitations.

The feedback-loop implementation of the pose estimation and controller modules necessitates either an initial bounding box or pose prediction for initialisation. We practically solve this by allowing a 'running start' of the experiment. We initiate the pushing using MoCap object poses, after which the pushing controller can start planning and predicting, which initialises the pose estimator. There are various ways to mitigate this limitation. One simple yet effective proposal is to start each pushing trial with the box placed within a tight range of possible locations relative to the robot, which means that a fixed bounding box can be used for initialisation of the pose estimator. A more intricate (yet admittedly ironic) solution is to use a 2D detector to only supply the initial bounding box. Even though this adds efforts that we wish to circumvent, it carries additional potential. One advantage is the pushing of multiple objects, since the 2D detector can simultaneously perform object classification and 2D localisation. Once the object class has been detected, the accompanying pre-trained weights are loaded for pose estimation and object pushing dynamics, after which the pushing can be started.

5.2 Limitations

Our proposed method does not support recognising previously unknown objects. Moreover, it requires a comprehensive dataset and CAD model of the object for training the vision pipeline. In addition, the inherent delay caused by the pose estimator's computation time forces the pushing controller to work with estimations from the past, resulting in a drop in pose prediction accuracy, which may limit real-time responsiveness or pushing performance decrease on similar or other objects. By using more appropriate hardware, such as a desktop CPU and dedicated GPU, the runtime of the computer vision module can be further reduced, as is reported in [39]. Judging the results from experiment 2, we argue that a faster runtime will lead to improved prediction accuracy, thus reducing the discrepancy between MoCap-based and pose estimator-based pushing. Since our implementation is modular, a faster and/or more accurate 2D detector based pose estimator can be used to substitute the implemented ZebraPose estimator.

5.3 Failures

While our method successfully performs object pose estimation in real-time in the vast majority of experiments, we do observe cases of instability. In some cases of instability, the pipeline recovers and continues the pushing. In other cases, the pipeline failed. We present visualisations of failure cases in Appendix C. We observe that the *Kalman Filter* causes instability, which occurs during the start-up phase of the pose estimator. When the pose estimator's Kalman Filter initialises on the first pose estimates, it occasionally outputs extreme values for the object state vector. We observe that this behaviour doesn't cause issues for the MoCap-based pushing, since that Kalman Filter has already stabilised on the 120Hz stream of pose data by the time the Planning and Control stage has initialised. If the resulting instability progresses into the Planning and Control stage, three outcomes are possible: 1) The *lost-contact* detection that checks whether contact is lost between robot and object is activated. By design, the controller stops the pipeline. 2) future state predictions based on the faulty state vector lead to an *RoI* which doesn't contain the object, leading to RANSAC/PnP failure and subsequent pipeline failure. 3) future state predictions based on the faulty state vector lead to a bounding box which doesn't lie in the image plane, which corrupts the calculation of the *RoI*. This causes a pipeline failure. This instability occurred in a limited portion of trials (3 out of 35).

This instability was partially mitigated by allowing the pose estimator-based *Kalman Filter* to stabilise on a certain amount of pose estimates, before the Planning and Control stage is allowed to use the state estimates.

We believe that this issue will be limited to our hybrid MoCap/pose estimator implementation, since a more advanced implementation including robot ego-pose estimation and an initial object detection stage might naturally mitigate or eliminate this issue, since it will require the pose to be estimated before pushing can begin, which allows the Kalman Filter to stabilise early on, as opposed to a mid-push initialisation of the Kalman Filter.

6 Conclusions

This work introduces a framework for simultaneous object pose estimation and object pushing in real-time. By leveraging the temporal coherence of robotic object pushing scenarios, we significantly reduce the computation time (>50x speedup) of the 2D detector that is commonly used in object pose estimation. This is achieved by using state predictions from an MPPI-based pushing controller to find the future location of the object in the image frame. We empirically show that the reduction of runtime of the 2D detector does not limit the pose estimator's accuracy. Additionally, it is demonstrated that the pose estimator's speedup is an essential step towards replacing the Motion Capture systems in object pushing methods. Finally, real-world experiments are performed to evaluate the pushing performance of our proposed framework and we compare it to a baseline method. We demonstrate that real-time object pushing using an accelerated object pose estimation method is achievable. Finally, we introduce the *AIRlab* dataset specifically for computer vision-guided planar object pushing.

Acknowledgments

I would like to express my gratitude to all those who played a role in the execution and completion of this master thesis.

To my parents, Wendy and Paul, and my sister, Noortje, your patience, emotional (and financial) support, as well as guarding my graduation process has been an invaluable throughout my studies. To Annefloor, for the guidance and support, which has allowed me to overcome obstacles and remain focused on my goals.

Furthermore, I extend my appreciation to Yujie, my daily supervisor, and Martijn, my supervisor, for your expert guidance and valuable feedback, especially at the times they were most needed. Your support has helped shape the direction of my research, refine my ideas and finalise my thesis.

The study-related and non-related skirmishing and decompressing with my friends, both those who have left Delft some time ago and my fellow-graduates that were close-by to embark on this path with me, have been invaluable.

Thank you all for being part of this milestone, and for giving me the support and willpower to **push** forward, especially when the tunnel seemed to be never-ending.

REFERENCES

[1] Tony Baltovski. husky_control, Nov. 2022. https://github.com/husky/husky.

[2] Tony Baltovski. mocap_optitrack, Nov. 2022. https://github.com/ros-drivers/mocap_optitrack.

[3] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing, Sept. 2017. arXiv:1704.03033 [cs, stat].

[4] Yannick Bukschat and Marcus Vetter. EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach. arXiv:2011.04307 [cs], Nov. 2020. arXiv: 2011.04307.

[5] Hansheng Chen, Pichao Wang, Fan Wang, Wei Tian, Lu Xiong, and Hao Li. EPro-PnP: Generalized End-to-End Probabilistic Perspective-n-Points for Monocular Object Pose Estimation. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Mar. 2022. arXiv: 2203.13254.

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation, Dec. 2017. arXiv:1706.05587 [cs].

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models, Nov. 2018. arXiv:1805.12114 [cs, stat].

[8] Lin Cong, Hongzhuo Liang, Philipp Ruppel, Yunlei Shi, Michael Görner, Norman Hendrich, and Jianwei Zhang. Reinforcement Learning With Vision-Proprioception Model for Robot Planar Pushing. Frontiers in Neurorobotics, 16, 2022.

[9] Xinke Deng, Arsalan Mousavian, Yu Xiang, Fei Xia, Timothy Bretl, and Dieter Fox. PoseRBPF: A Rao–Blackwellized Particle Filter for 6-D Object Pose Tracking. IEEE Transactions on Robotics, 37(5):1328–1342, Oct. 2021. Conference Name: IEEE Transactions on Robotics.

[10] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1), 2022.

[11] Zhaoxin Fan, Yazhi Zhu, Yulin He, Qi Sun, Hongyan Liu, and Jun He. Deep Learning on Monocular Object Pose Detection and Tracking: A Comprehensive Overview. ACM Computing Surveys, page 3524496, Mar. 2022.

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, June 1981.

[13] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In Kyoung Mu Lee, Yasuyuki Matsushita, James M. Rehg, and Zhanyi Hu, editors, Computer Vision – ACCV 2012, Lecture Notes in Computer Science, pages 548–562, Berlin, Heidelberg, 2013. Springer.

[14] Tomas Hodan, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Fabian Manhardt, Federico Tombari, Tae-Kyun Kim, Jiri Matas, and Carsten Rother. BOP: Benchmark for 6D Object Pose Estimation, Aug. 2018. arXiv:1808.08319 [cs].

[15] Francois R Hogan and Alberto Rodriguez. Reactive planar non-prehensile manipulation with hybrid model predictive control. The International Journal of Robotics Research, 39(7):755–773, June 2020. Publisher: SAGE Publications Ltd STM.

[16] Tingbo Hou, Adel Ahmadyan, Liangkai Zhang, Jianing Wei, and Matthias Grundmann. MobilePose: Real-Time Pose Estimation for Unseen Objects with Weak Shape Supervision. arXiv:2003.03522 [cs], Mar. 2020. arXiv: 2003.03522.

[17] Y. Hu, L. Wang, J. Liang, and T. Wang. Cooperative box-pushing with multiple autonomous robotic fish in underwater environment. IET Control Theory &amp; Applications, 5(17):2015–2022, Nov. 2011. Publisher: IET Digital Library.

[18] Shun Iwase, Xingyu Liu, Rawal Khirodkar, Rio Yokota, and Kris M. Kitani. RePOSE: Fast 6D Object Pose Refinement via Deep Texture Rendering. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), pages 3283–3292, Oct. 2021.

[19] Alexander Jung. imgaug, 2020. https://github.com/aleju/imgaug.

[20] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. 2017 IEEE International Conference on Computer Vision (ICCV), pages 1530–1538, Oct. 2017.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, Dec. 2014.

[22] Earl J. Kirkland. Bilinear Interpolation. In Earl J. Kirkland, editor, Advanced Computing in Electron Microscopy, pages 261–263. Springer US, Boston, MA, 2010.

[23] Senka Krivic and Justus Piater. Pushing corridors for delivering unknown objects with a mobile robot. Autonomous Robots, 43(6):1435–1452, Aug. 2019.

[24] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. CosyPose Consistent Multi-view Multi-object 6D Pose Estimation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, Proceedings of the European Conference on Computer Vision 2020 (ECCV), Lecture Notes in Computer Science, pages 574–591, Cham, Aug. 2020. Springer International Publishing.

[25] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep Iterative Matching for 6D Pose Estimation. In Proceedings of the European Conference on Computer Vision (ECCV), pages 683–698, 2018.

[26] Zhigang Li, Gu Wang, and Xiangyang Ji. CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 7678–7687, 2019.

[27] Walter Lucetti and Aymeric Dujardin. zed_ros_wrapper, Nov. 2022. https://github.com/stereolabs/zed-ros-wrapper/.

[28] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. Methods for Non-Linear Least Squares Problems (2nd ed.). Technical University Denmark, 2004.

[29] Tekin Meriçli, Manuela Veloso, and H. Levent Akın. A Case-Based Approach to Mobile Push-Manipulation. Journal of Intelligent & Robotic Systems, 80(1):189–203, Dec. 2015.

[30] Pedro Nogueira. Motion capture fundamentals. In Doctoral Symposium in Informatics Engineering, volume 303, 2011.

[31] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation. In Proceedings of the European Conference on Computer Vision (ECCV), pages 119–134, 2018.

[32] Kiru Park, Timothy Patten, and Markus Vincze. Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation. In Proceedings of the IEEE/CVF International Conference on Computer Vision 2019 (ICCV), pages 7668–7677, 2019.

[33] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4561–4570, June 2019.

[34] Susan Potters. Learning-based control for pushing with a non-holonomic mobile robot. Master's thesis, Delft University of Technology, Delft, Nov. 2022.

[35] Mahdi Rad and Vincent Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 3848–3856, Oct. 2017.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Advances in Neural Information Processing Systems, volume 28. Curran Associates, Inc., 2015.

[37] Chen Song, Jiaru Song, and Qixing Huang. HybridPose: 6D Object Pose Estimation Under Hybrid Representations. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 428–437, June 2020. ISSN: 2575-7075.

[38] Jochen Stüber, Claudio Zito, and Rustam Stolkin. Let's Push Things Forward: A Survey on Robot Pushing. Frontiers in Robotics and AI, 7, 2020.

[39] Yongzhi Su, Mahdi Saleh, Torben Fetzer, Jason Rambach, Nassir Navab, Benjamin Busam, Didier Stricker, and Federico Tombari. ZebraPose: Coarse to Fine Surface Encoding for 6DoF Object Pose Estimation. arXiv:2203.09418 [cs], Mar. 2022. arXiv: 2203.09418.

[40] Sebastian Thrun. Probabilistic robotics. Communications of the ACM, 45(3):52–57, 2002. Publisher: ACM New York, NY,

USA.

[41] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully Convolutional One-Stage Object Detection. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 9626–9635, Oct. 2019.

[42] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. Conference on Robot Learning (CoRL) 2018, Sept. 2018. arXiv: 1809.10790.

[43] P.M. Van der Burg. BOP-ROS conversion toolkit, Nov. 2022. https://github.com/ThijsvdBurg/Husky_scripts.

[44] P.M. Van der Burg. BOP Toolkit, Nov. 2022. https://github.com/ThijsvdBurg/bop_toolkit.

[45] P.M. Van der Burg. object_pose_estimation_and_pushing, Nov. 2022. https://github.com/ThijsvdBurg/object_pose_estimation_and_pushing.

[46] P.M. Van der Burg. ZebraPose, Oct. 2022. https://github.com/ThijsvdBurg/ZebraPose.

[47] Bowen Wen, Chaitanya Mitash, Baozhang Ren, and Kostas E. Bekris. se(3)-TrackNet: Data-driven 6D Pose Tracking by Calibrating Image Residuals in Synthetic Domains. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 10367–10373, Oct. 2020. arXiv: 2007.13866.

[48] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. arXiv:1711.00199 [cs], May 2018. arXiv: 1711.00199.

[49] Yan Xu, Kwan-Yee Lin, Guofeng Zhang, Xiaogang Wang, and Hongsheng Li. RNNPose: Recurrent 6-DoF Object Pose Refinement With Robust Correspondence Field Estimation and Pose Optimization. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 14880–14890, 2022.

[50] Zisong Xu, Rafael Papallas, and Mehmet Dogar. Real-Time Physics-Based Object Pose Tracking during Non-Prehensile Manipulation, Mar. 2023. arXiv:2211.13572 [cs].

[51] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. A high-fidelity experimental dataset of planar pushing. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 30–37, Oct. 2016. ISSN: 2153-0866.

[52] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the Continuity of Rotation Representations in Neural Networks. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 5738–5746, Long Beach, CA, USA, June 2019. IEEE.

[53] Hai Zhu and Javier Alonso-Mora. Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments. IEEE Robotics and Automation Letters, 4(2):776–783, Apr. 2019. Conference Name: IEEE Robotics and Automation Letters.

APPENDIX A
PUSHING PARAMETERS OF REAL-WORLD EXPERIMENTS

Parameter	Value
MPPI prediction horizon H	20
MPPI sampled trajectories U	150
MPPI object predictive model (PNN) ensemble size	3
MPPI obstacle avoidance	False
MPPI update frequency	10 Hz
MoCap update frequency	120 Hz
RANSAC/PnP iterations	150
ZebraPose update frequency	7.5[2]
RANSAC/PnP initial pose guess	False
Number of pose estimates before switching from MoCap to Pose Estimator pushing	6

TABLE 5. Parameters of real-world experiments using our proposed framework. We cover all notable MPPI controller, Motion Capture and ZebraPose inference parameters. training the ZebraPose pose estimator on the *AIRlab* training split. [2]Depending on the RANSAC/PnP iterations. A pose estimator frequency of 8.5 Hz real-time inference could be achieved using 25 RANSAC/PnP iterations.

APPENDIX B
SCHEMATIC OVERVIEW OF ROS SOFTWARE FRAMEWORK

Here the ROS software framework is visualised schematically. We extend the ROS framework from [34] and we modify the `mppi_control` (Planning and Control stage from Section 3.1) to supply an object state prediction to the `pose_to_bbox`, which is the 2D detector stage (explained in Section 3.2).

We present a list of packages used and their respective sources in Table 6.

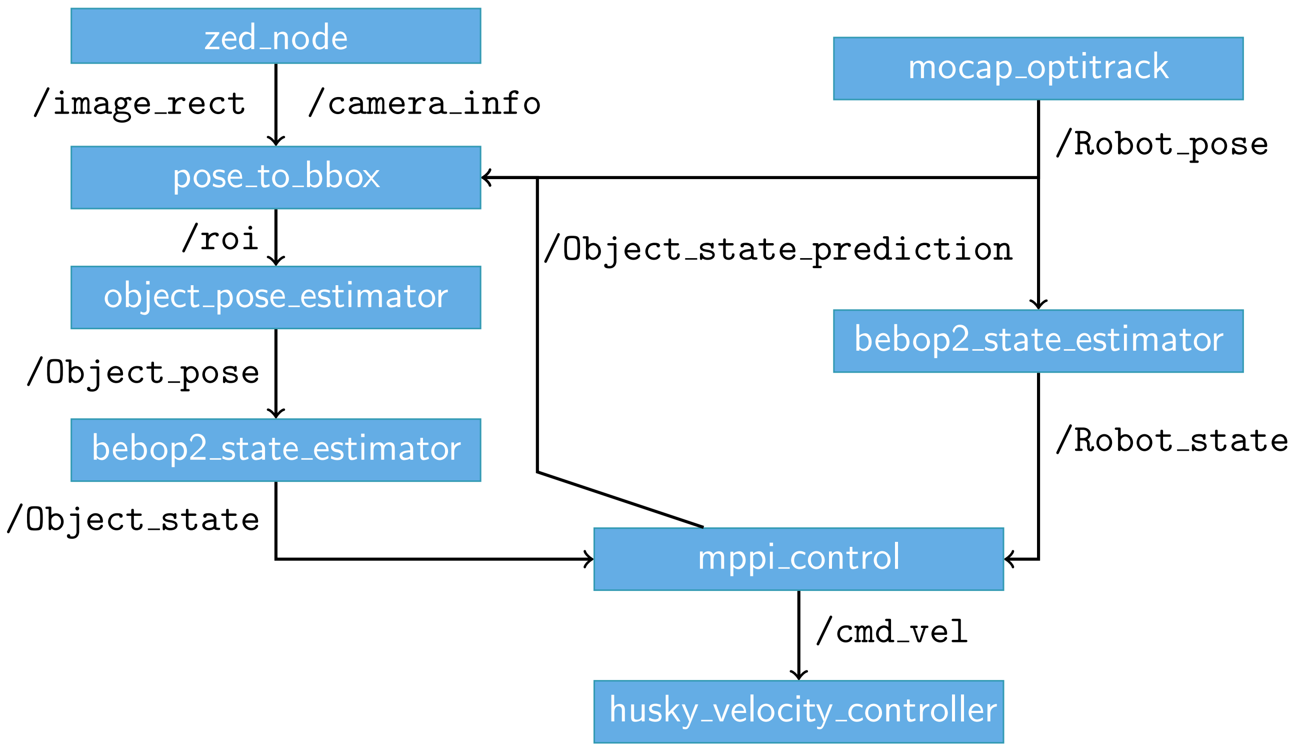


Fig. 10. A schematic overview of the most important ROS packages used for our proposed pipeline.

Package	Source or author	Comment
mocap_optitrack	[2]	[-]
bebop2_state_estimator	[52]	Modified from [34] to extend it to pose estimates
mppi_control	[34]	Modified from [34] to supply pose predictions
zed_node	[27]	[-]
pose_to_bbox	P.M. van der Burg [45]	[-]
object_pose_estimator	P.M. van der Burg[45]	The ROS package was developed in this work. We take the ZebraPose [39] inference code to get a pose from an *RoI*. RANSAC/PnP code was modified to allow initial pose guess
husky_velocity_controller	[1]	[-]

TABLE 6. List of ROS packages used for our framework, along with the source to allow credits where they are due.

Appendix C
Pipeline failure cases

Here we visualise the cases of pipeline failure using our method. As discussed in Section 5, the initialisation of the pose estimator-based state estimator can cause a large error to occur in the initial state estimates. This results in a large *PNN* prediction error which causes the pipeline to fail. We show the pre- and post failure RGB images and `rviz TF display` snapshot.

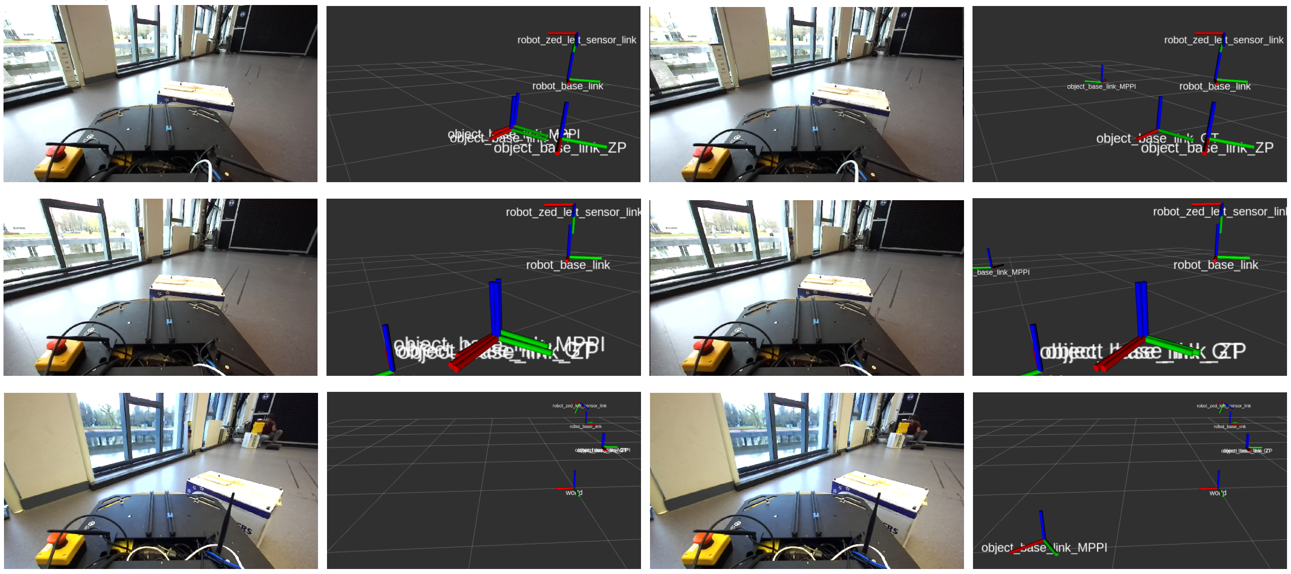


Fig. 11. Visualisations of failure cases. For each case the moment before (left) and after (right) failure is shown. Each row in total. We show the camera's RGB image and an RVIZ tf visualisation of the most important coordinate frames. The common cause of failure is that the state estimator suddenly estimates the location of the object at a faraway location, which is reflected by the subsequent MPPI prediction.

APPENDIX D
DATA AUGMENTATION

The RGB input images are augmented during training to help the CNN generalise to changes in illumination, motion blur, unseen occlusions etc. We list the data augmentation methods in Table 7. All augmentations are inherited from the `imgaug` codebase [19], specifically the `augmenters` package. 80% of the images are sent through the augmentation sequence, after which a per-augmentation probability applies. The Invert augmentation has an additional probability set.

Augmentation	Probability	Parameter 1 + value	Parameter 2 + value
General[3]	0.8	[-]	[-]
SaltAndPepper	0.3	fraction of pixels=0.05	[-]
MotionBlur	0.2	kernel_size=5	[-]
CourseDropout	0.5	p=0.2	size_percent=0.05
GaussianBlur	0.5	range=1.2*U[0,1)	[-]
Add	0.5	range=(-25,25)	per_channel=0.3
Invert	0.3	Secondary Probability=0.2	per_channel=True
Multiply	0.5	range=(0.6,1.4)	per_channel=0.5
LinearContrast	0.5	range=(0.5,2.2)	per_channel=0.3

TABLE 7. Following [39], the training input image is augmented in various ways using the `imgaug` codebase, specifically the `augmenters` package. [3]General probability of image subject to augmentations.