External Camera-Based Robot Pose Estimation Method

Short proposal

Full name: Nabil Miri

Matriculation number: 243299

Introduction

- **Pose Estimation**

Determining the position and orientation in an environment.

- **Uses:**

Navigation – Manipulation - ....

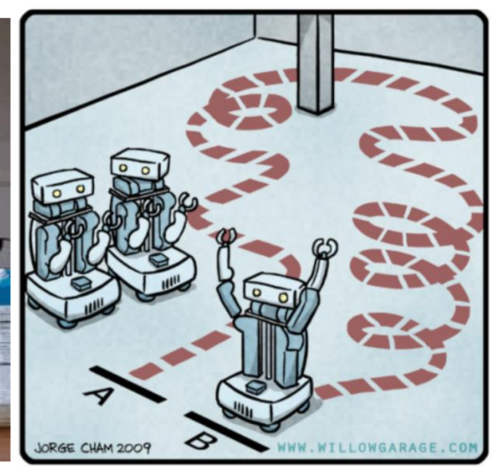
- **Different Approaches:**

Satellite-based Localization - Network-based Localization - Indoor localization – Visual Localization etc…

⇒**Interest:** Visual Localization

No ‘best’ method

R.O.B.O.T. Comics



"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Paper 1: Camera-to-Robot Pose Estimation from a Single Image [Timothy E. Lee et al. (Dec 2019) NVIDIA + CMU]

- Main Contributions of this paper:

- External Camera Pose Estimation from a **single** RGB image using Deep neural network to detect 2D **keypoints**

- Trained the network **only** on **synthetic** data

- Used **PnP** to estimate the camera to robot transformation

- Creating an Online Calibration Method

1.1.1 Approaches: Classical Approach

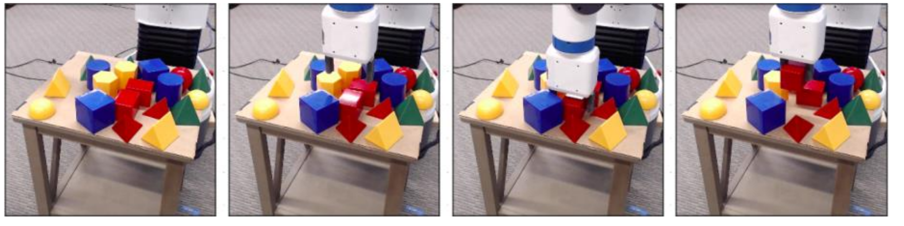
- Fiducial **markers** at the end effector => **Collect** several images =>Solve a homogenous linear equation system for the transformation

- **Cumbersome** procedure of physically modifying the end effector to collect a set of images, running an off-line calibration procedure, and (optionally) removing the fiducial marker.

- The entire calibration procedure must be **repeated** from scratch if the camera moves.

1.1.2 Approaches: Recent Approaches

- Using **deep learning** to map RGB images to world coordinates on a table.
- In this approach the learned mapping is **specific** to the task or environment.
- Preventing the mapping from being applied to new tasks or environments without **retraining**.



1.2 Procedure:

- **E**ncoder-**d**ecoder neural network processes the image to produce a set of n **belief maps**, one per **keypoint**.

- Then, **PnP** uses the 2D belief maps, along with the forward kinematics and the camera intrinsics, to compute the camera-to-robot pose, ${}^{R}_{C}T$.

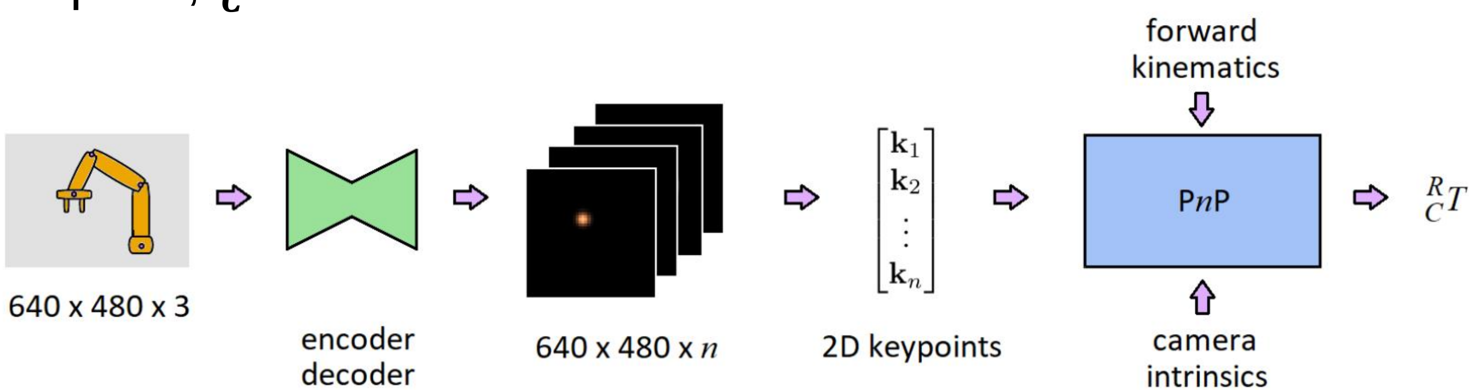
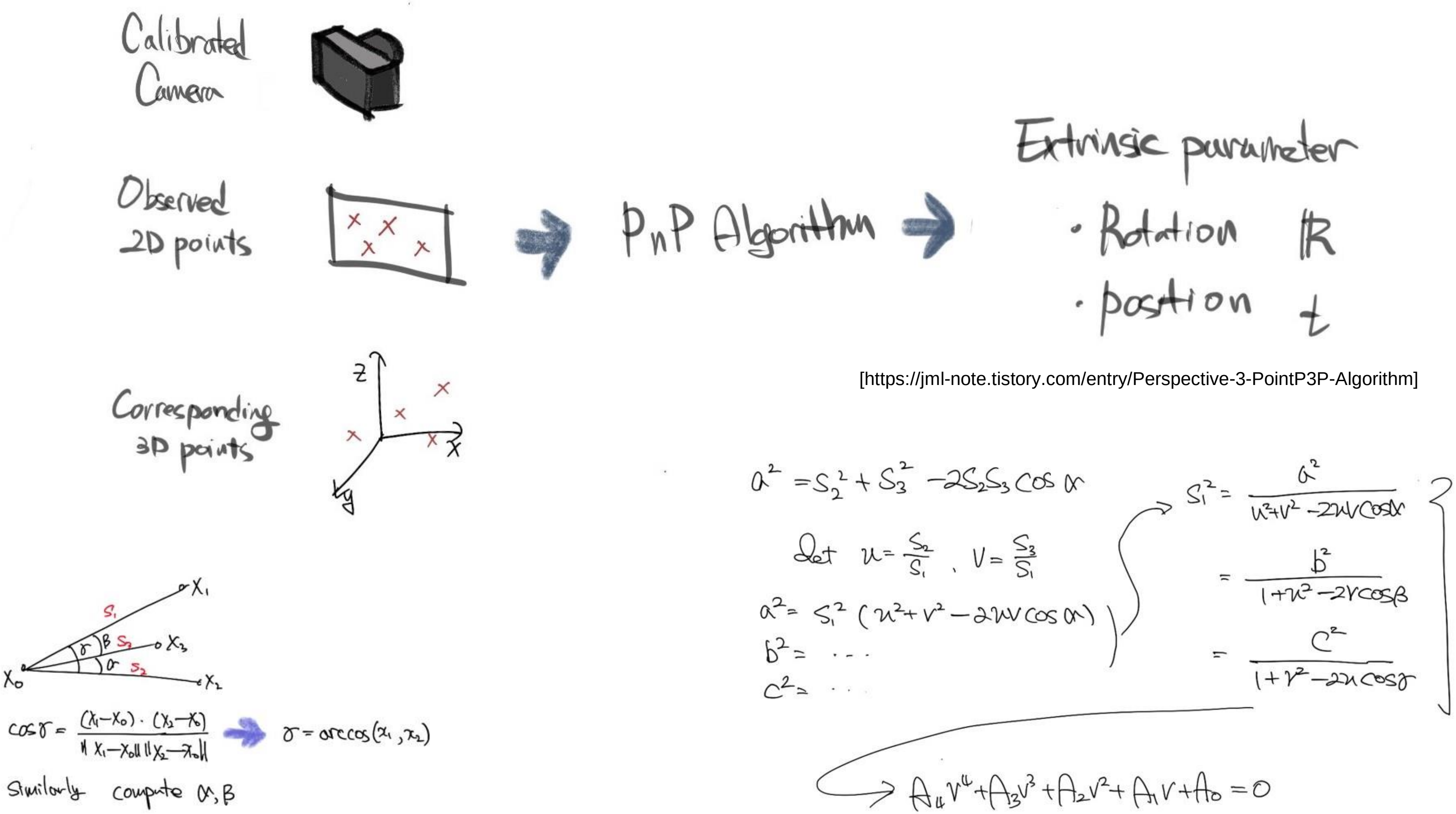


Fig. 1. The DREAM framework. A deep encoder-decoder neural network takes as input an RGB image of the robot from an externally-mounted camera, and it outputs n belief maps (one per keypoint). The 2D peak of each belief map is then extracted and used by PnP, along with the forward kinematics and camera intrinsics, to estimate the camera-to-robot pose, ${}^{R}_{C}T$.

Perspective-n-Point (PnP):

The goal of the Perspective-n-Point problem (PnP) is to find the relative pose between an object and a camera from a set of n pairings between 3D points and their corresponding 2D projections



[https://jml-note.tistory.com/entry/Perspective-3-PointP3P-Algorithm]

1.2.1 Procedure: Network Architecture:

- Encoder-Decoder network to detect the **keypoints** - takes as **input** an RGB image of size.

- The output captures a **2D belief map** for each keypoint, where pixel values represent the likelihood that the keypoint is projected onto that pixel.

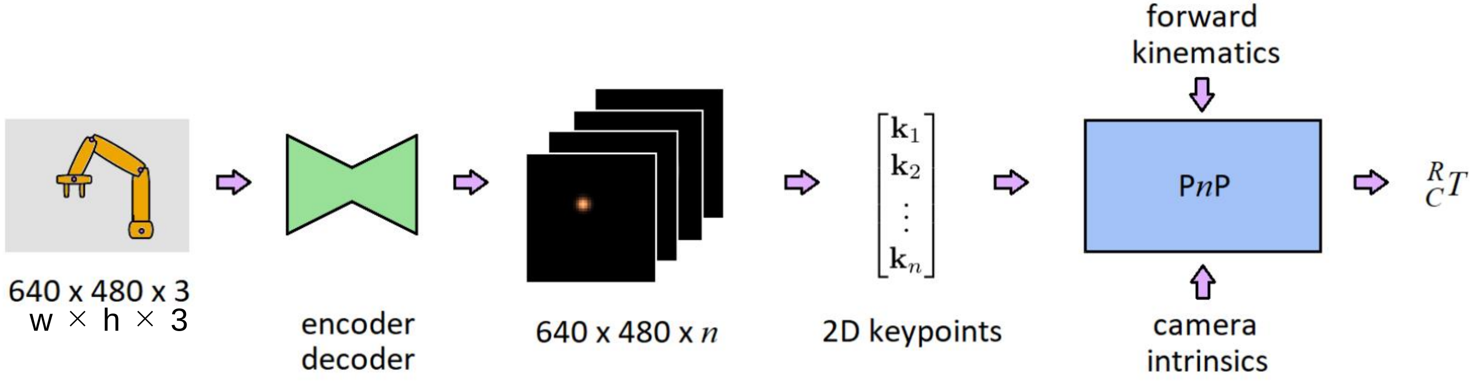
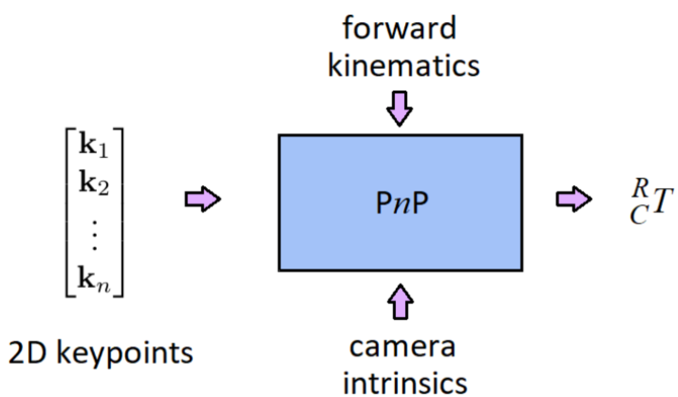
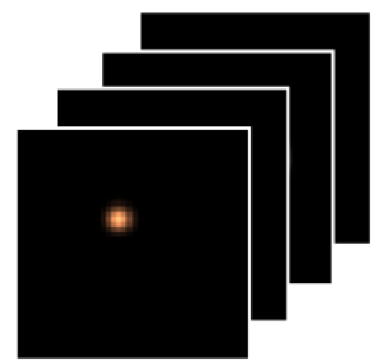


Fig. 1. The DREAM framework. A deep encoder-decoder neural network takes as input an RGB image of the robot from an externally-mounted camera, and it outputs n belief maps (one per keypoint). The 2D peak of each belief map is then extracted and used by PnP, along with the forward kinematics and camera intrinsics, to estimate the camera-to-robot pose, $^R_C T$.

1.2.2 Procedure: Pose Estimation

- Given the 2D keypoint coordinates, robot joint configuration with forward kinematics, and camera intrinsics, **PnP** is used to retrieve the pose of the robot
- The **keypoint coordinates** are calculated as a **weighted average** of values near thresholded peaks in the output belief maps.
- Applying Gaussian smoothing to the belief maps to reduce the effects of noise.

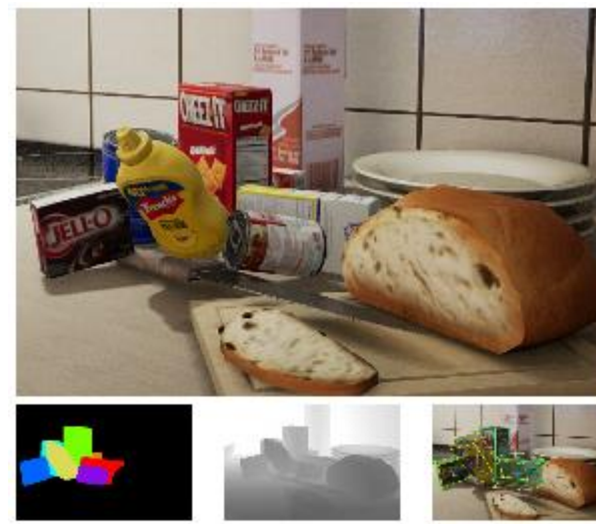




1.2.3 Procedure: Data Generation

- Network is trained using **only** synthetic data with domain randomization (**DR**) and image augmentation

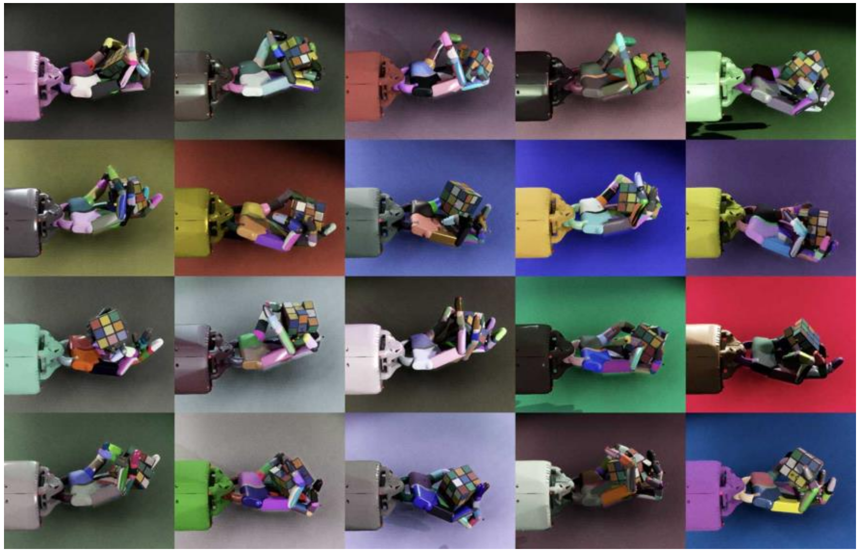
- To generate the data, open-source NVIDIA Deep learning Dataset Synthesizer (**NDDS**) tool was used, which is a plugin for the UE4 game engine. NDDS is augmented to export **2D/3D keypoint locations**



Domain Randomization:

- Domain Randomization Domain randomization is a systematic approach to data generation process that aims to enhance generalization of the machine learning algorithms to new environments.

Deep Learning with Domain Randomization



OpenAI Shadow Arm to solve Rubik's Cube

1.2.3 Procedure: Data Generation

- Various randomizations were applied:

1) The robot's **joint angles** were randomized within the joint limits.

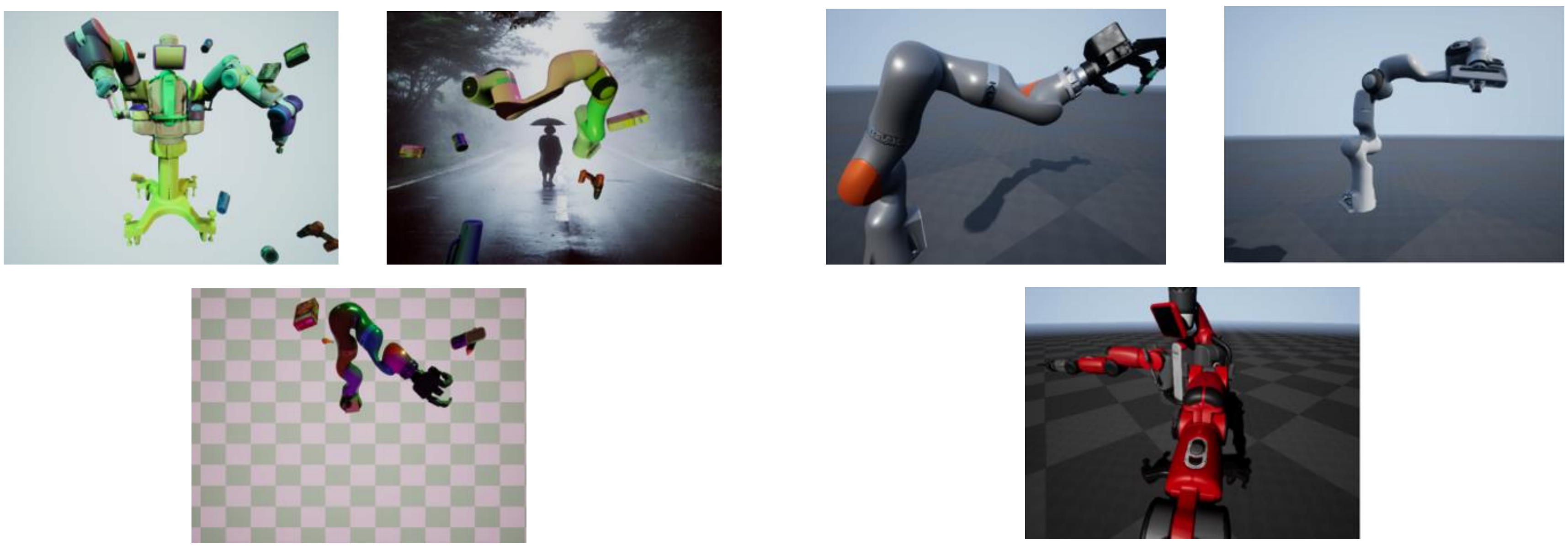
2) The **camera** was positioned freely in a somewhat truncated hemispherical shell around the robot, with angle ranging from -135° to $+135^\circ$

3) Three scene **lights** were positioned and oriented freely while randomizing both intensity and color.

4) The scene **background** was randomly selected from the COCO dataset.

1.2.3 Procedure: Data Generation

- Data Generation:



domain-randomized (DR) non-DR

Fig. 2. Synthetic training images for the three robot models: Franka Panda (top), Kuka LBR with Allegro hand (middle), and Rethink Baxter (bottom).

1.3 Metrics:

- Experiments were done on 3D data and the metric used:

- The **Average Distance (ADD)** Metric:
  - Average Euclidean distance of all 3D keypoints to their transformed versions, using the estimated camera pose as the transform.
  - ADD is a principled way to combine rotation and translation errors

$$ADD = \frac{1}{n} \sum_{i=1}^{n} ||\widetilde{\mathbf{T}}_b^c \overline{\mathbf{p}}_i - \mathbf{T}_b^c \overline{\mathbf{p}}_i||_2$$

T : is the ground-truth camera-to-robot pose
Tˆ: is the estimated camera-to-robot pose
P : is the 3D keypoint location
n : is the number of points

1.3 Metrics:

- 3 versions of DREAM network were used:
  - VGG or Resnet Encoder
  - **F**ull – **H**alf – **Q**uarter decoder output resolutions

- Configuration:
  - 50 epochs
  - Adam Optimization Algorithm
  - 1.5e-4 learning rate
  - 0.9 momentum
  - 100K synthetic DR images

1.3 Metrics:

- A pose estimation is considered correct if ADD (-S) is **smaller** than an average distance **threshold.**
- The improvement due to **increasing** resolution is clear, but different architectures have only minimal impact for most scenarios.

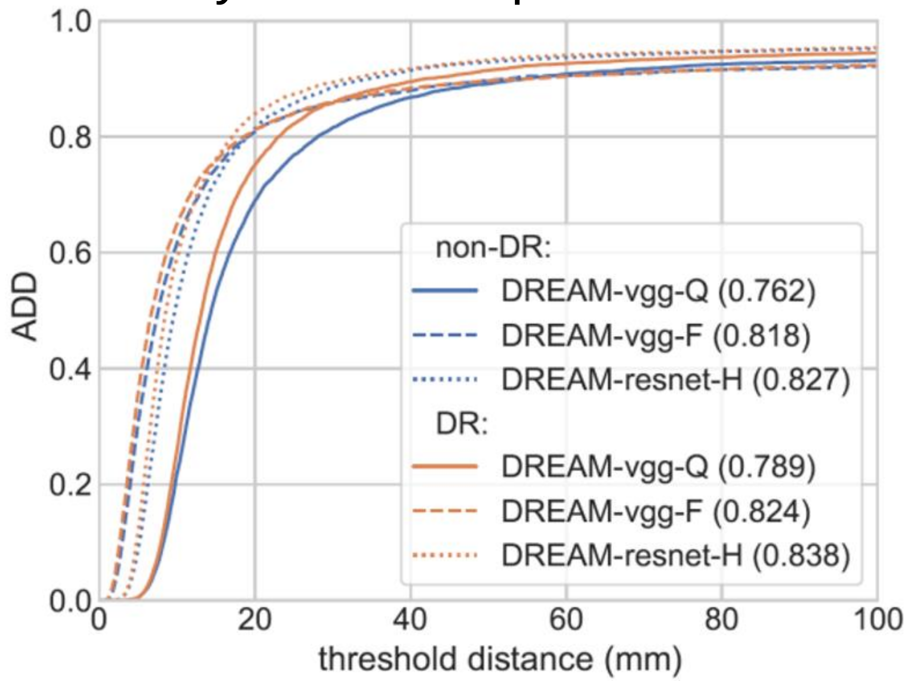


Fig. 3. ADD results for three different variants of DREAM network on the **simulated** datasets. The numbers in parentheses are the area under the curve (AUC).

1.3 Metrics:

- These results show that the training procedure is able to **bridge the reality gap**: There is only a modest difference between the best performing network on **simulated** and **real data**.

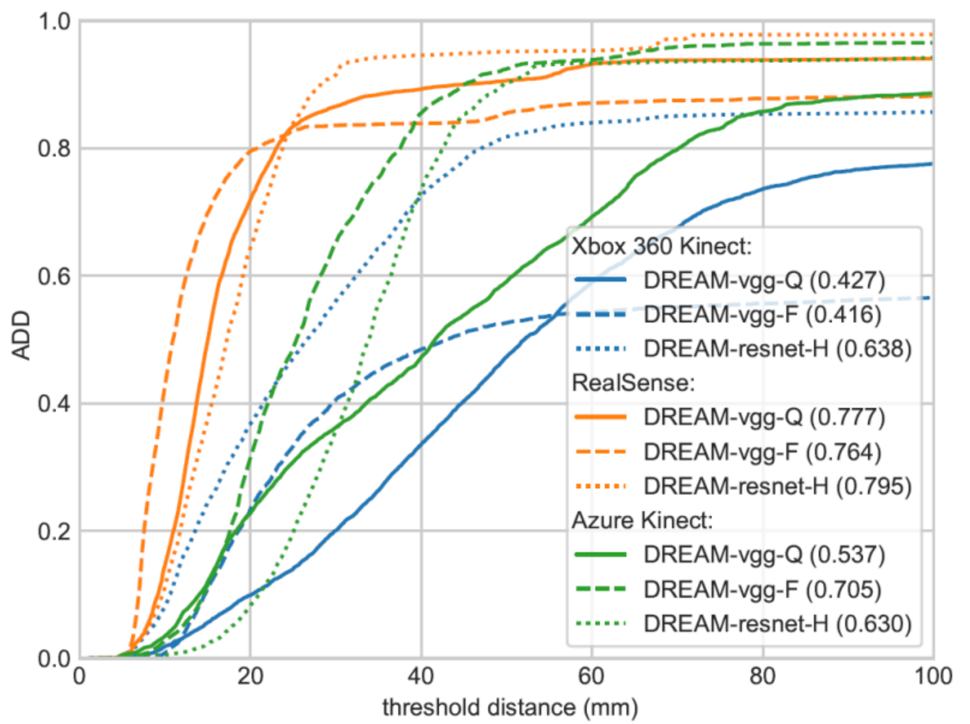


Fig. 4. ADD results on the **real** Panda-3Cam dataset.

Paper 2: Markerless Camera-to-Robot Pose Estimation via Self-supervised Sim-to-Real Transfer

[Jingpei Lu, Florian Richter, and Michael C. Yip University of California, San Diego (Mar 2023)]

- End-to-end pose estimation framework that is capable of online camera-to-robot **calibration** and a **self-supervised** training method to scale the training to unlabeled real-world data
- Approaches to robot pose estimation are classified into two categories: **keypoint-based** and **rendering-based** methods
- The CtRNet uses keypoints for **faster** inference speed and rendering for **higher** performance for image-level self-supervision is used.

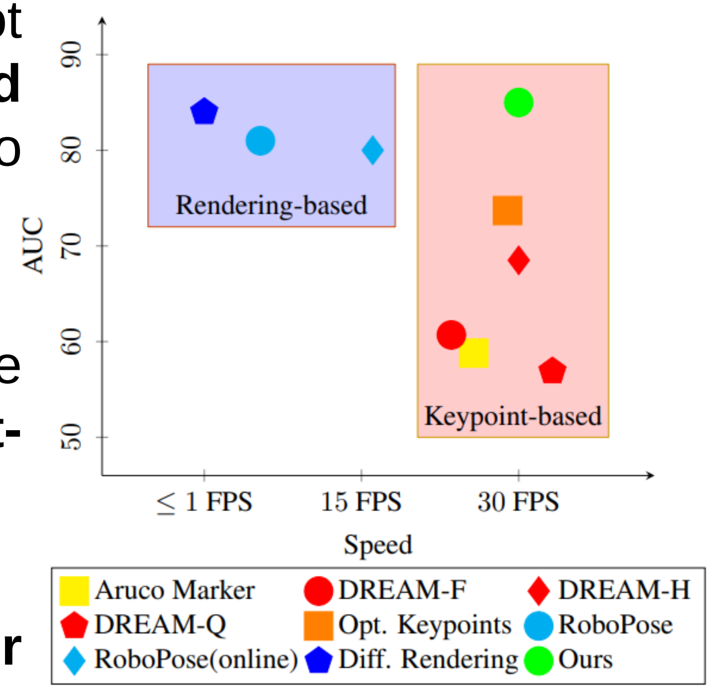
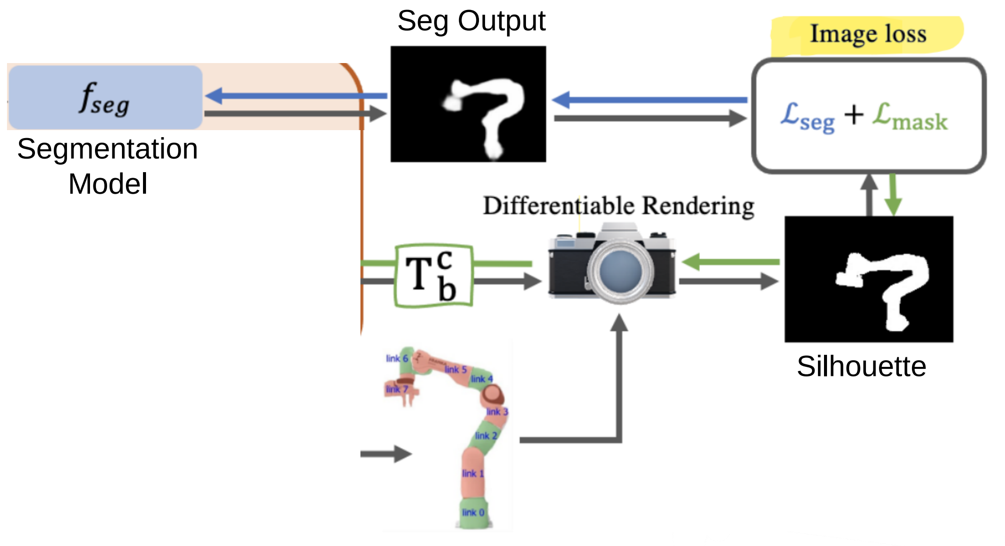


Figure. Comparison of speed and accuracy (based on AUC metric) for existing image-based robot pose estimation methods.

Paper 2: Markerless Camera-to-Robot Pose Estimation via Self-supervised Sim-to-Real Transfer

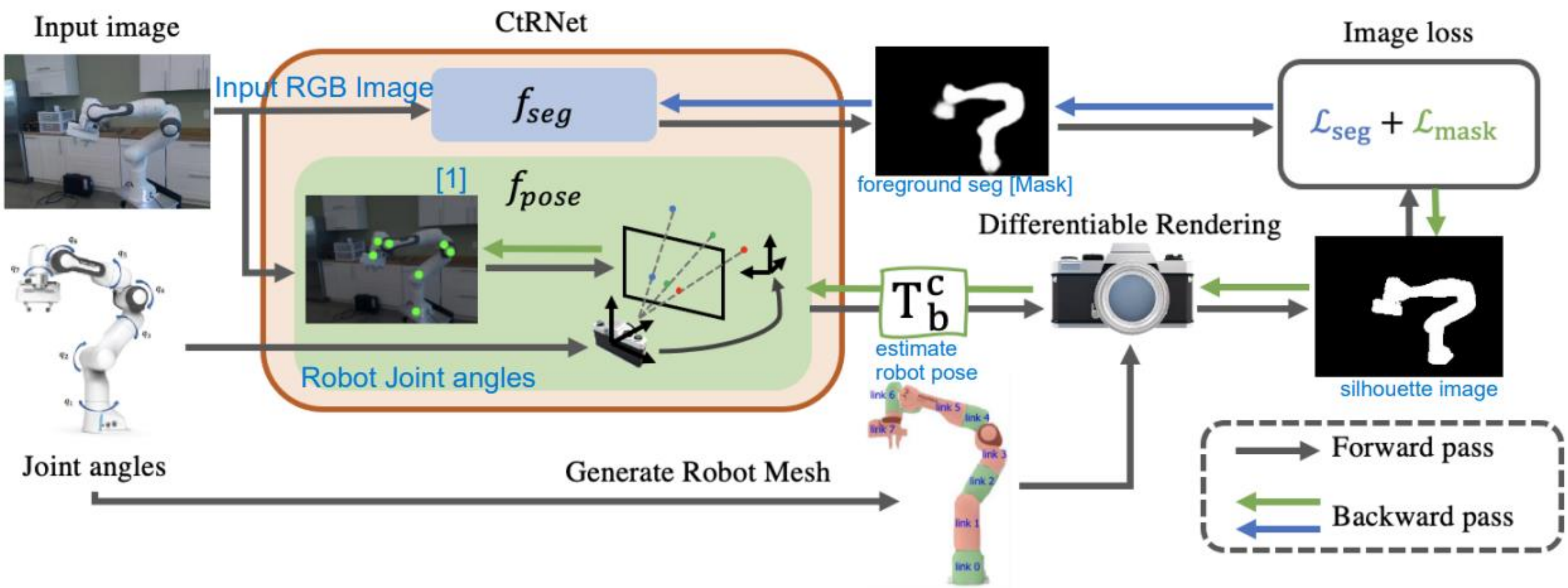
- **Differential Rendering:**



Source: DFR: Differentiable Function Rendering for Learning 3DGeneration from Images

2.1: Self-Supervised Training for Sim-to-Real Transfer

- Most **effective** way to adapt the neural network to the real world is directly training the network on **real** data.
- **Self supervised** method train network without **3D annotations**.
- Pipeline includes **foreground segmentation** to generate a mask of the robot, fseg, alongside the **pose estimation**, fpose.



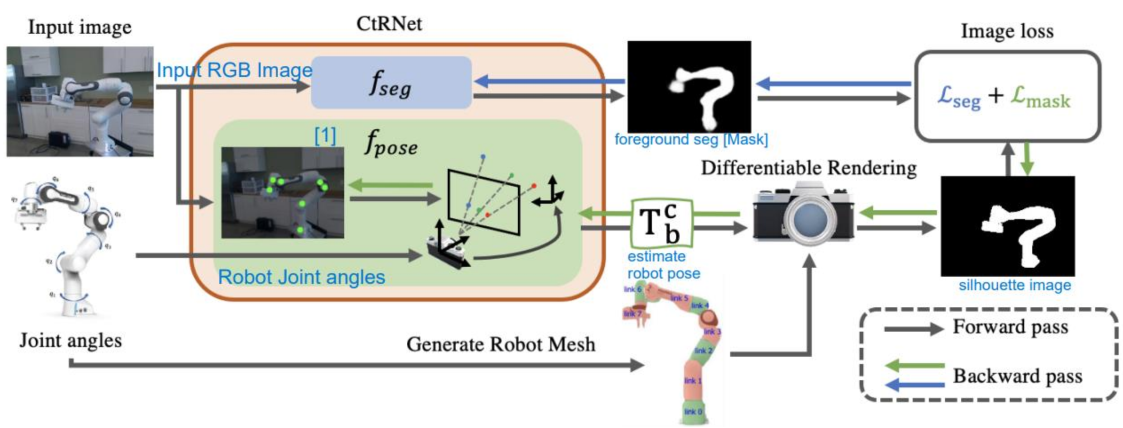
2.1: Self-Supervised Training for Sim-to-Real Transfer

- The self-supervised **objective:** optimize neural network parameters by minimizing the difference between the rendered silhouette image and the mask image.

$$\theta_{bb}, \theta_{kp}, \theta_{seg} = \underset{\theta_{bb},\theta_{kp},\theta_{seg}}{\arg\min}\ \mathcal{L}[f_{seg}(\mathbb{I}|\theta_{bb}, \theta_{seg}),$$
$$\mathcal{R}(f_{pose}(\mathbb{I}|\mathbf{q}, \theta_{bb}, \theta_{kp})|\mathbf{K})]$$

L: loss function capturing the image differences
R: diff renderer
K: camera params

Bb: backbone
Kp: keypoint
I: RGB Image



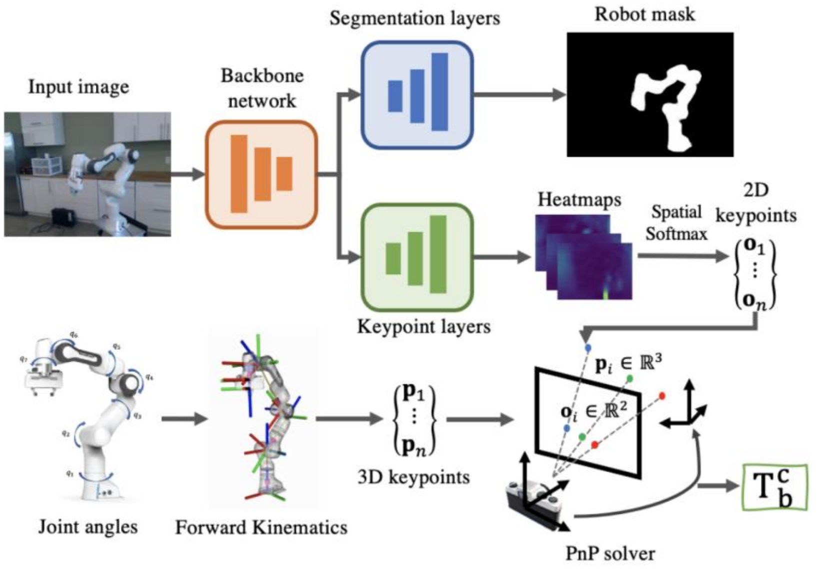
2.1: Self-Supervised Training for Sim-to-Real Transfer

- CtRNet's parameters (fseg and fpose) pretrained, with **synthetic** data.
- During the **self-training phase**, where CtRNet learns with **real** data, the objective loss captures the difference between the segmentation result and the rendered image.
- The loss is **iteratively back-propagated** to, Θ, where each iteration **fseg** and **fpose** take turns learning from each other to overcome the sim-to-real gap.

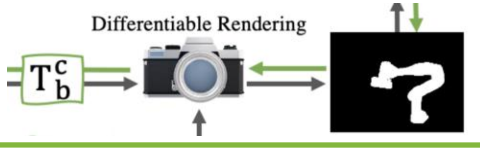
$$\mathbb{M} = f_{seg}(\mathbb{I}|\theta_{bb}, \theta_{seg})$$

M: robot mask

$$\mathbf{T}_b^c = f_{pose}(\mathbb{I}|\mathbf{q}, \theta_{bb}, \theta_{kp})$$



- Silhouette from **Differential Ren.**



2.1: Self-Supervised Training for Sim-to-Real Transfer

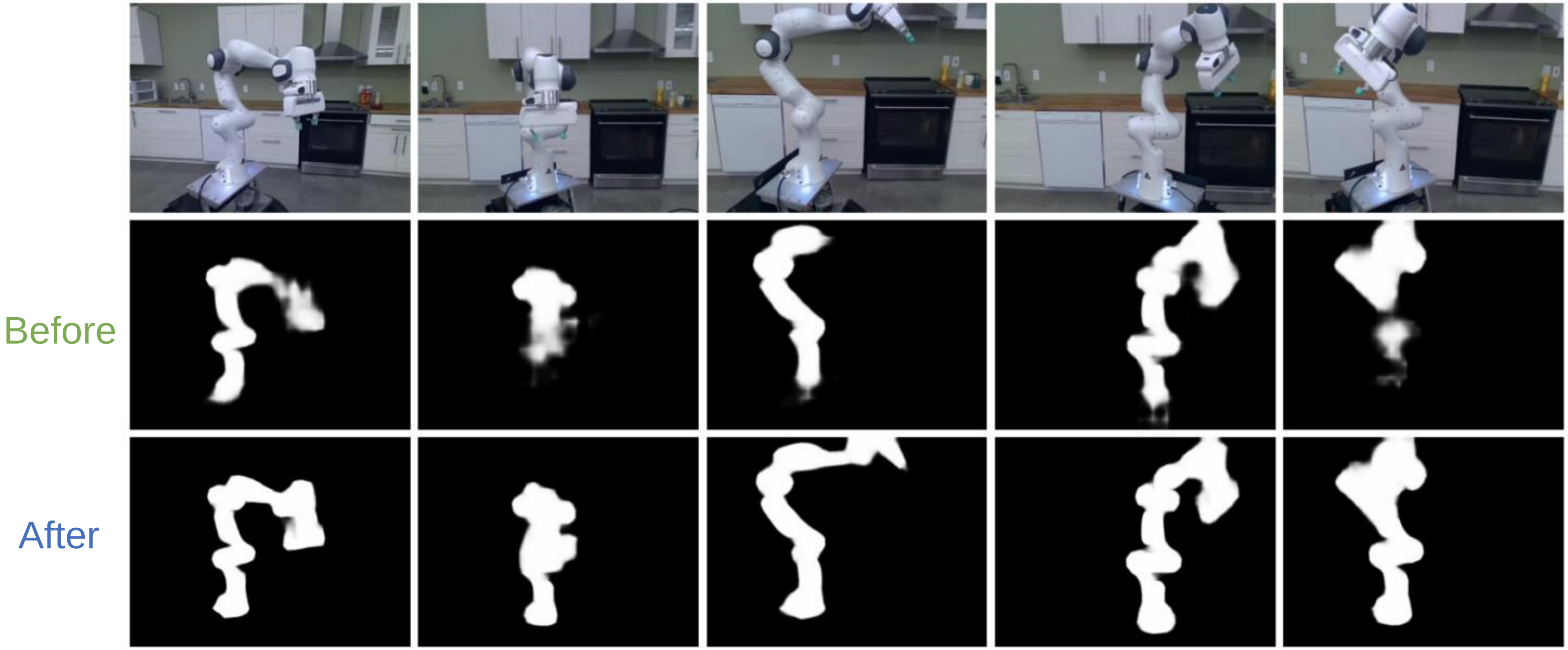


Figure 9. Qualitative results of CtRNet foreground segmentation before and after self-supervised training. From top to bottom row shows the RGB images, the segmentation masks before, and after self-supervised training.

2.2 Experiments:

- DREAM-real Dataset: Real-world robot dataset collected with 3 different cameras: Azure Kinect, XBOX 360 Kinect, and RealSense.
- Contains 50K RGB images

Method	Category	Backbone	All	
			AUC ↑	Mean (m) ↓
DREAM-F [29]	Keypoint	VGG19	60.740	113.029
DREAM-Q [29]	Keypoint	VGG19	56.988	59.284
DREAM-H [29]	Keypoint	ResNet101	68.584	17.477
CtRNet	Keypoint	ResNet50	**85.962**	**0.020**

Comparison of paper's methods with the state-of-the-art methods on DREAM-real datasets using ADD metric.

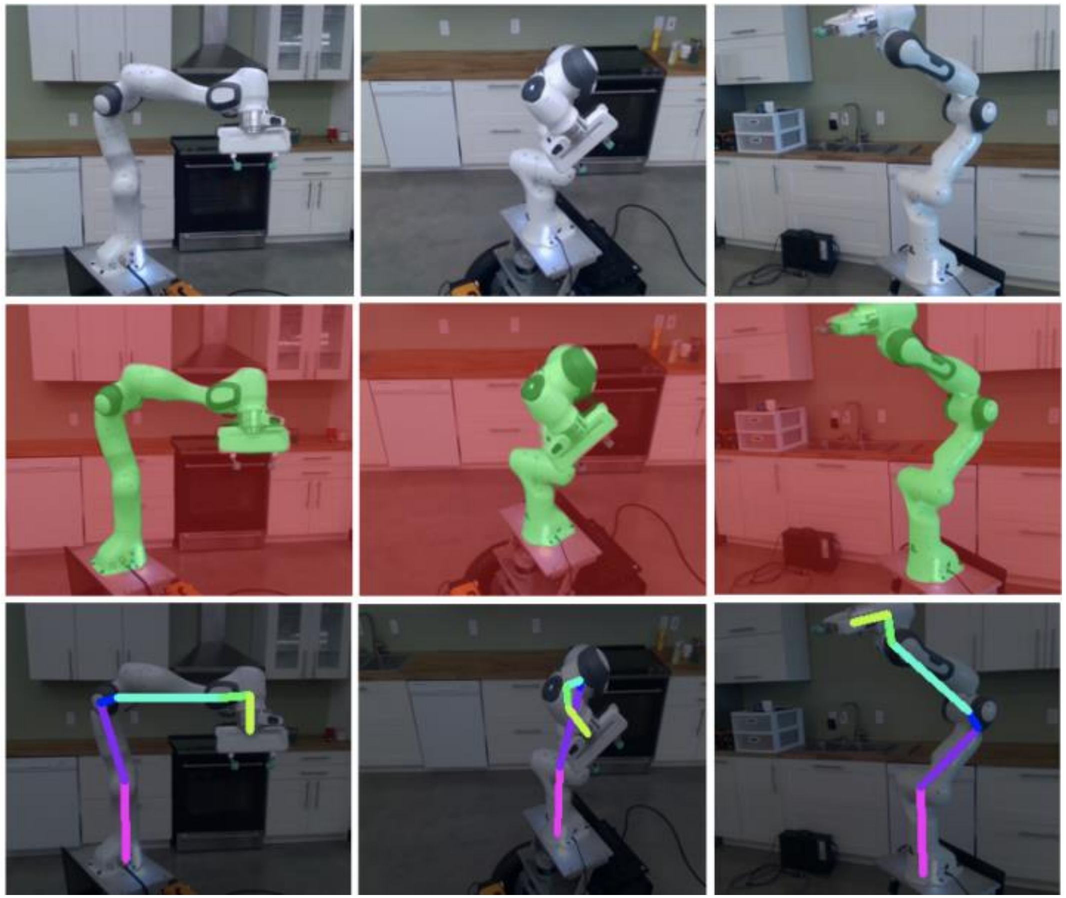
2.2 Experiments:

- Qualitative results of CtRNet foreground segmentation and pose estimation on DREAM-real dataset

Input RGB Image:

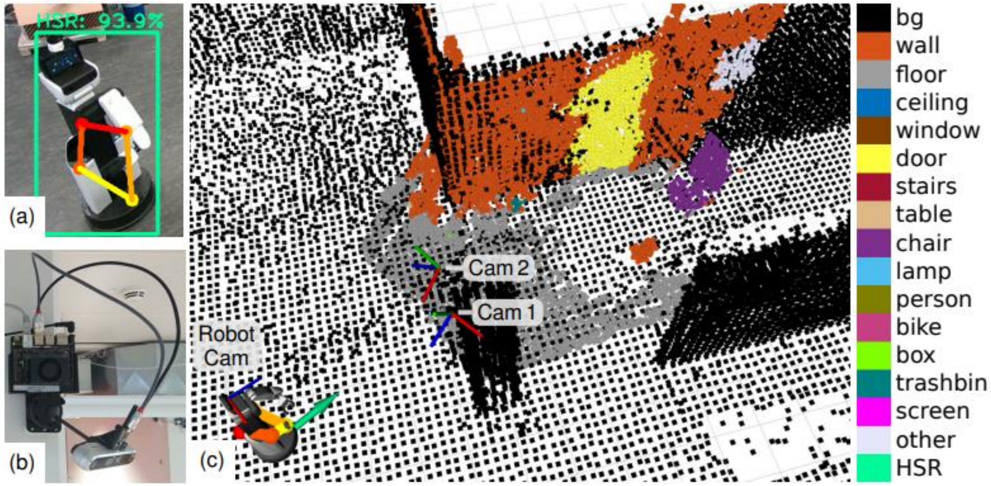
Foreground Segmentation:

Projected skeleton based on estimated robot pose:



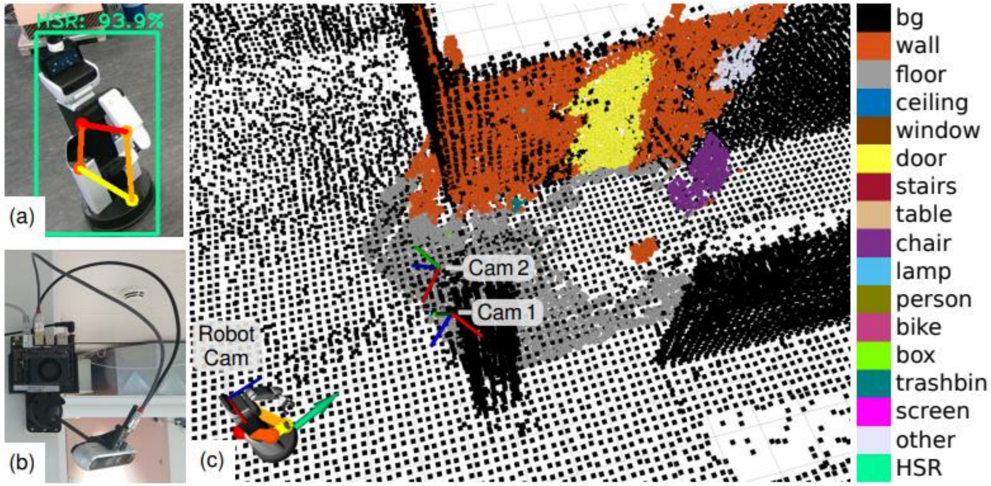
Paper 3: External Camera-based Mobile Robot Pose Estimation for Collaborative Perception with Smart Edge Sensors

- **Multi-view** keypoint detection marker-less mobile robot pose estimation
- **Collaborative perception** in real-world scenes between mobile robot and sensor network to build a globally consistent **3D semantic map**.
- For robot pose estimation, Convolutional Neural Networks (CNNs) used for robot **detection** and estimation of 2D **keypoints**



Paper 3: External Camera-based Mobile Robot Pose Estimation for Collaborative Perception with Smart Edge Sensors

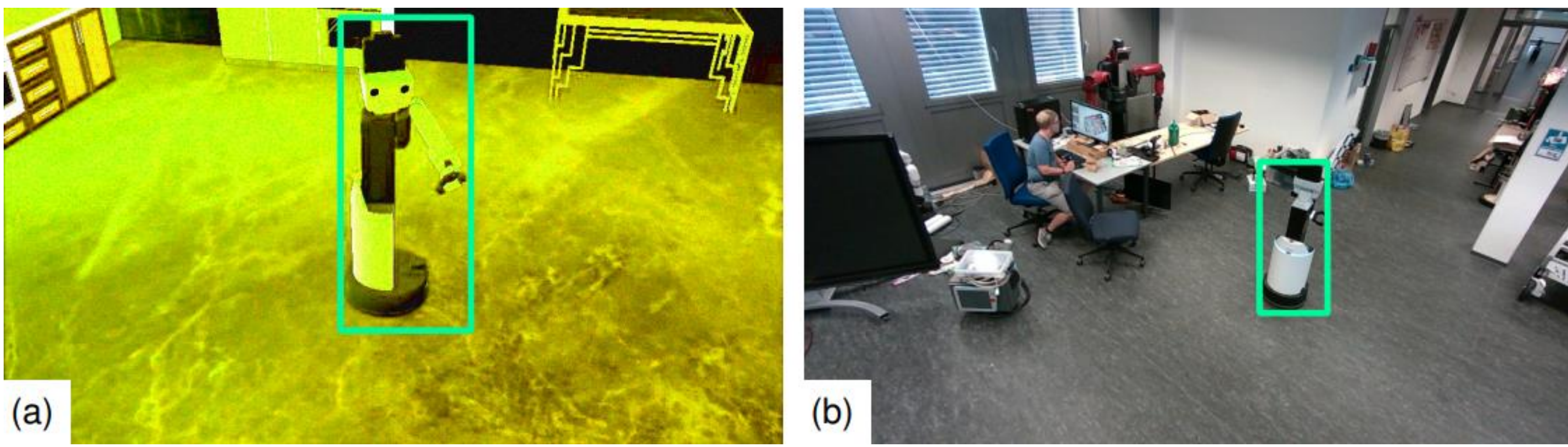
- CNN for keypoint estimation is trained only on **synthetic** data obtained through **randomized** scene generation
- Robot keypoint detections are used to estimate the robot's pose via **multi-view minimization of reprojection errors**.
- Multiple sources for **localization**, the external camera views + robot's internal 2D LiDAR-based navigation, **increases** robustness in highly cluttered, dynamic real-world environments, where few distinct features, such as walls or columns, are not visible in the LiDAR due to **occlusions**



[Simon Bultmann, Raphael Memmesheimer, and Sven Behnke (uni of Bonn) ICRA June 23]

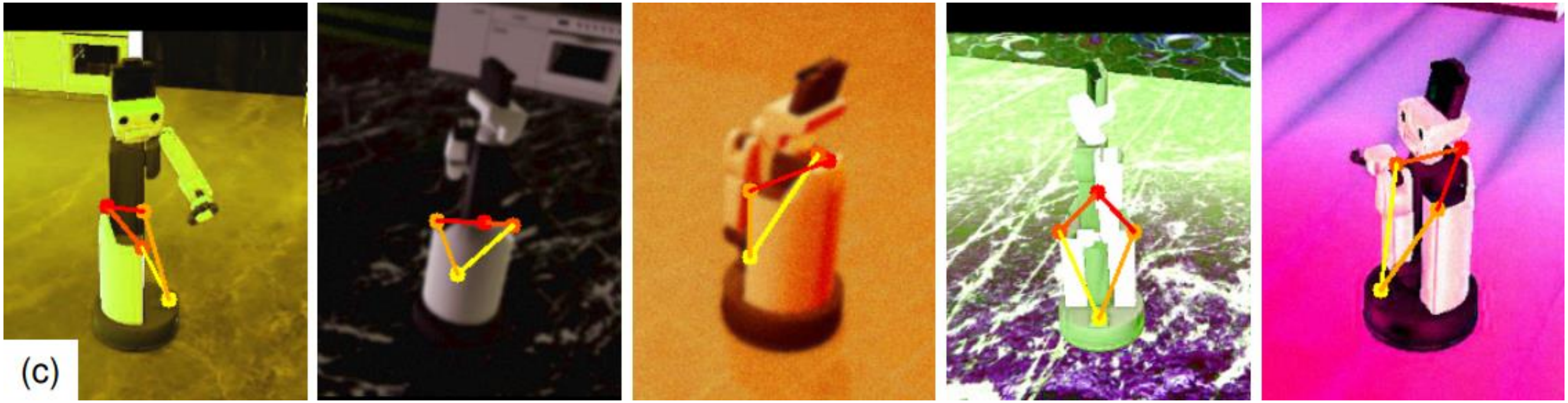
3.1 Method: A. Robot Keypoint Detection

- 1) Network Architecture:
  - Detecting a **b**ounding **box** of the robot
  - Estimating **keypoints** on the **crop** of the robot
  - MobileDet architecture is used for robot detection and a network with a MobileNet V3 backbone for keypoint estimation.



3.1 Method: A. Robot Keypoint Detection

- 2) Training Data:
  - Train the networks predominantly on **synthetic** data
  - CNN for **keypoint** estimation is trained purely on *simulated* data (36k samples), while we combine *synthetic* data and *manually* annotated real images (12k resp. 3.5k samples) for robot **detection**
  - The combination of *real and synthetic* data helps to boost detector performance in *highly cluttered* real-world environments



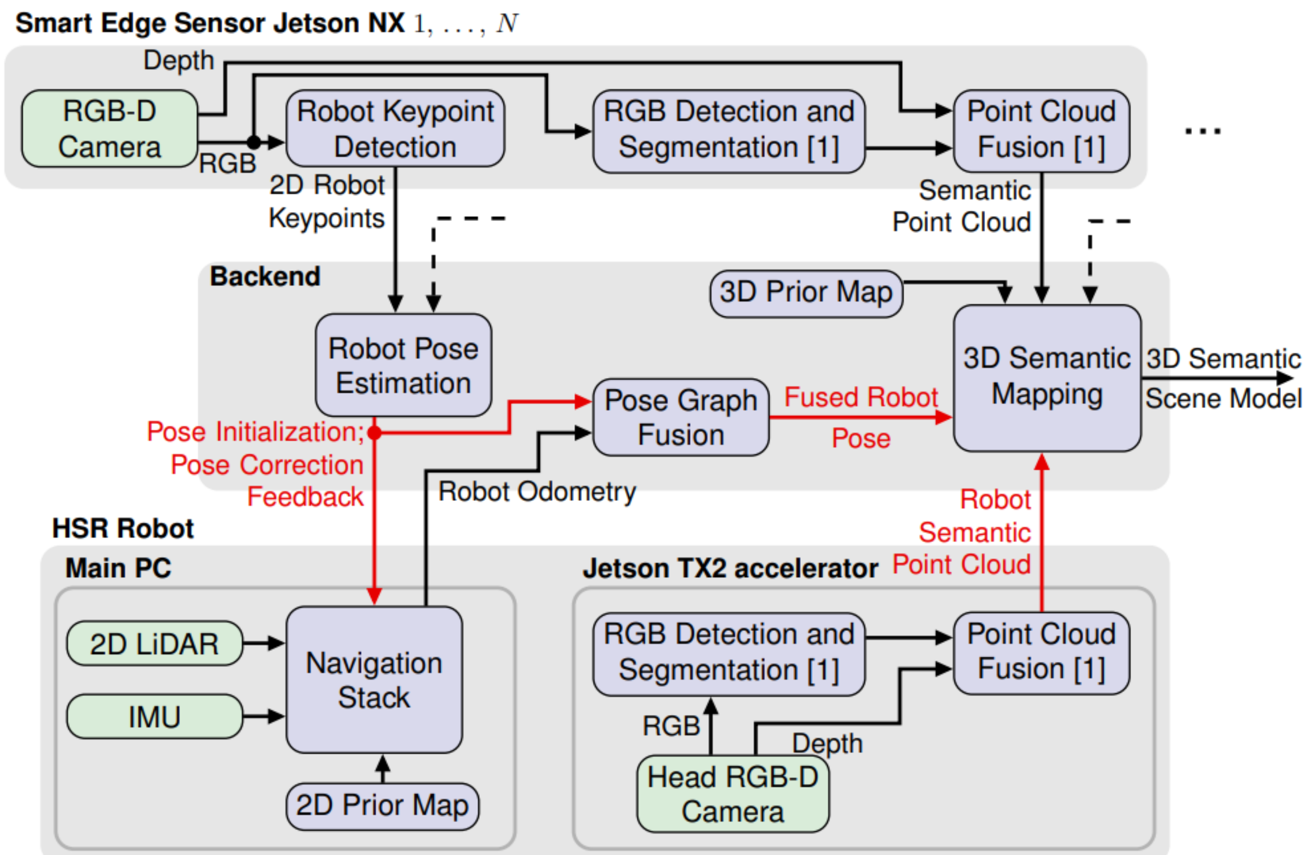
3.1 Method: B. Robot Pose Estimation

- 2D robot **keypoints** are sent over a network to a central backend, where detections from **multiple** cameras are synchronized. The robot pose ${}^{W}_{R}T$ is then recovered by solving a weighted nonlinear least squares problem via minimization of multi-view reprojection errors:

$$ {}^{W}_{R}\boldsymbol{T} = \underset{{}^{W}_{R}\boldsymbol{T}}{\arg\min} \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} \left\| \boldsymbol{k}_{ij} - \Pi_i \left({}^{C_i}_{W}\boldsymbol{T} \, {}^{W}_{R}\boldsymbol{T} \boldsymbol{p}_j \right) \right\|^2 $$

- N externally mounted cameras Ci , i = 1, . . . , N
- 2D projections $k_{ij} \in R^2$ of M keypoints $p_j \in R^3$, j = 1, . . . , M
- weights wij depending on the confidence of the keypoint detection in the respective camera
- Levenberg-Marquardt algorithm for optimization

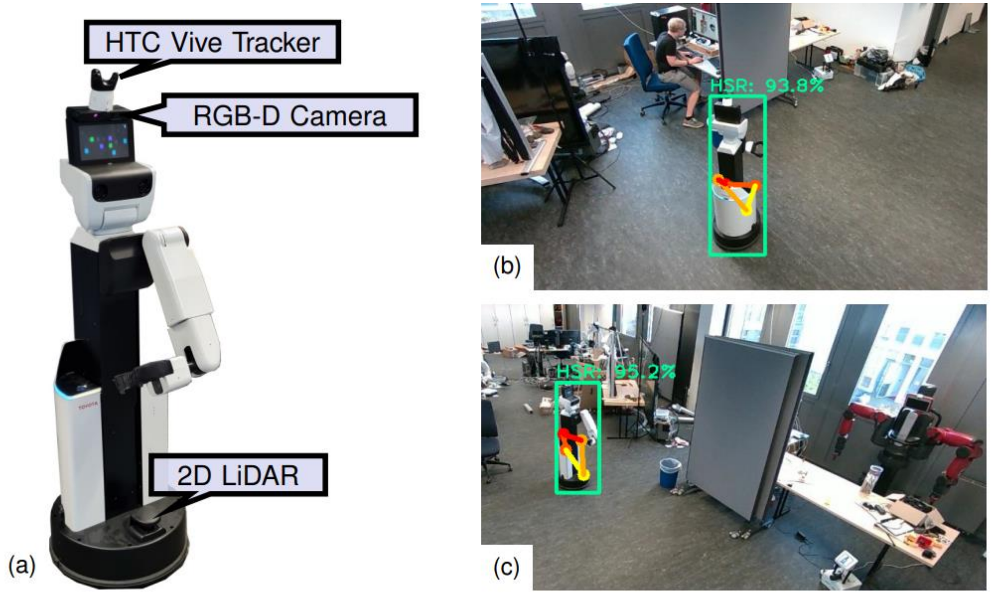
3.1 Method: B. Robot Pose Estimation



Overview of the proposed sensor network architecture for collaborative localization and perception. N external smart edge sensors observe mobile HSR robot and scene from static viewpoints. Robot pose is initialized and corrected via external camera pose estimation

3.2 EVALUATION:

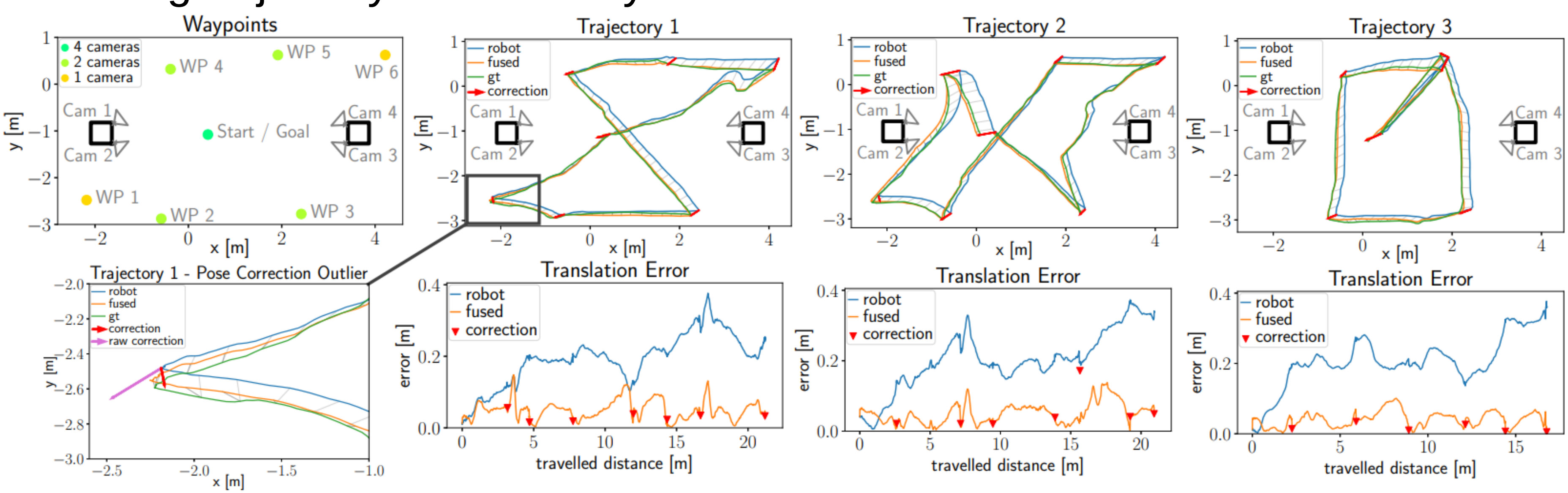
- **Four** external smart edge sensors are mounted at ~2.5 m height in the center of the room to initialize and correct the robot localization
- As a **reference** for pose estimation, **HTC Vive Pro tracking system** was employed which was shown to yield position accuracies within a few millimeters



- For evaluation of the pose estimation accuracy, we define **seven waypoints** in the area observed by the external cameras and connect them in different ways to **three** different trajectories

3.2 EVALUATION:

- The **pose correction** from external cameras is **not** sent to the robot during these experiments to measure the deviation over the full trajectory.
- Using **only** the internal LiDAR-based localization, the robot cannot reach all waypoints in three of five trials and the success rate reaches only **1 / 5**. Failures occur after **38 m** of traveled distance, on average.
- With the proposed localization feedback, the robot completes the ~60 m long trajectory successfully in all trials.



3.2 EVALUATION:

- The typical **position error** of our **proposed method** for mobile robot pose estimation is below 2.8 cm when detected in at least **two** cameras, and below 4.3 cm when detected in a single camera, while the **robot** localization typically deviates more than 19 cm after only 5 m of traveled distance.

TRANSLATION AND ORIENTATION ERROR (MEAN ± STD) AT WAYPOINTS, BY NO. OF CAMERAS WITH ROBOT DETS. AND POSE ESTIMATION SOURCE.

Pose Estimation	1 Camera		2 Cameras		4 Cameras		**Average**	
Robot	20.6 ± 7.6 cm	1.17 ± 1.19°	17.1 ± 6.9 cm	1.03 ± 1.23°	25.0 ± 6.9 cm	1.30 ± 1.96°	19.1 ± 7.6 cm	1.11 ± 1.38°
Cameras (raw)	13.8 ± 10.1 cm	3.39 ± 2.87°	2.59 ± 1.49 cm	0.97 ± 0.79°	2.88 ± 1.42 cm	1.02 ± 0.76°	4.65 ± 6.22 cm	1.44 ± 1.72°
Cameras (1 frame)	8.23 ± 5.23 cm	1.40 ± 1.31°	2.59 ± 1.49 cm	0.97 ± 0.79°	2.88 ± 1.42 cm	1.02 ± 0.76°	3.65 ± 3.36 cm	1.06 ± 0.92°
Cameras (5 frames)	7.77 ± 5.47 cm	1.18 ± 1.27°	**2.58 ± 1.48 cm**	0.86 ± 0.68°	2.82 ± 1.43 cm	**0.97 ± 0.74°**	3.54 ± 3.31 cm	0.94 ± 0.84°
Fused	**4.25 ± 1.57 cm**	**1.12 ± 1.08°**	2.64 ± 1.48 cm	**0.79 ± 0.65°**	**2.73 ± 1.41 cm**	**0.97 ± 0.76°**	**2.93 ± 1.60 cm**	**0.88 ± 0.77°**

- The pose correction feedback **significantly** improves the **robot's localization**.

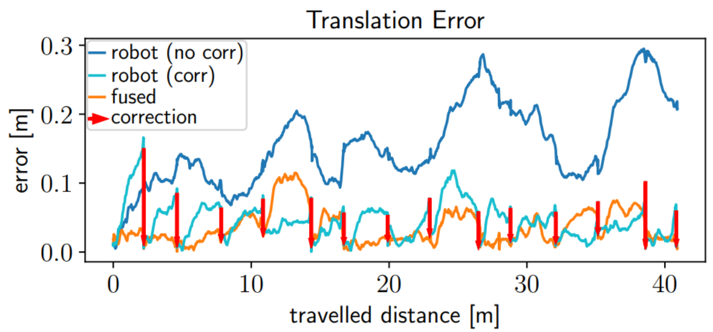


Fig. 6. Translation error with and without applying pose correction feedback to the robot's localization for the second experiment.

4 Final Thoughts:

- The majority of modern robotic automation utilizes **cameras** for sensory information about the environment to complete tasks and provide feedback for closed-loop control.

- **Markerless** camera-to-robot pose estimation which has been utilized for various **applications** such as surgical robotic manipulation and mobile robot manipulators etc.

- Continuous **Improvement** on these algorithms


THANKS