Keypoints-based Adaptive Visual Servoing for Control of Robotic Manipulators in Configuration Space

Sreejani Chatterjee[1], Abhay C. Karade [1], Abhinav Gandhi [1], and Berk Calli[1]

***Abstract*— This paper presents a visual servoing method for controlling a robot in the configuration space by purely using its natural features. We first created a data collection pipeline that uses camera intrinsics, extrinsics, and forward kinematics to generate 2D projections of a robot's joint locations (keypoints) in image space. Using this pipeline, we are able to collect large sets of real-robot data, which we use to train realtime keypoint detectors. The inferred keypoints from the trained model are used as control features in an adaptive visual servoing scheme that estimates, in runtime, the Jacobian relating the changes of the keypoints and joint velocities. We compared the 2D configuration control performance of this method to the skeleton-based visual servoing method (the only other algorithm for purely vision-based configuration space visual servoing), and demonstrated that the keypoints provide more robust and less noisy features, which result in better transient response. We also demonstrate the first vision-based 3D configuration space control results in the literature, and discuss its limitations. Our data collection pipeline is available at `https://github.com/JaniC-WPI/KPDataGenerator.git` which can be utilized to collect image datasets and train realtime keypoint detectors for various robots and environments.**

I. INTRODUCTION

Image-based Visual Servoing (IBVS) algorithms [1]–[3] allow to control robotic manipulators using image features and provide robust results even in the existence of camera and robot calibration errors. Identifying reliable visual features has a critical importance to these algorithms, since they constitute the control error. Thus far, for eye-to-hand applications, the visual servoing implementations in the literature rely on special feature-rich patterns or fiducial markers that are attached at the end effector of the robot [4]–[7]. While this approach is feasible in many robot control applications, it has limitations: It can only control the end effector pose of the robot, and cannot control the robot configuration, which is especially important for redundant robots. Moreover, the robot can be controlled only when the camera has a clear line of sight to the end effector markers. If the markers are occluded (either by self occlusion due to end effector pose or by other objects in the scene) or cannot be detected due to reflections or lighting conditions, the control operation may fail. Our research explores methods to control the robot via its natural features (keypoints) (Fig. 1). This brings several advantages: 1) we remove the dependency on placing artificial markers on the robot; 2) we enable the control of the robot in the configuration space by tracking multiple keypoints along its body; 3) our method is purely vision-based, i.e. it does not rely on the knowledge of robot Jacobian or joint encoder measurements in runtime.

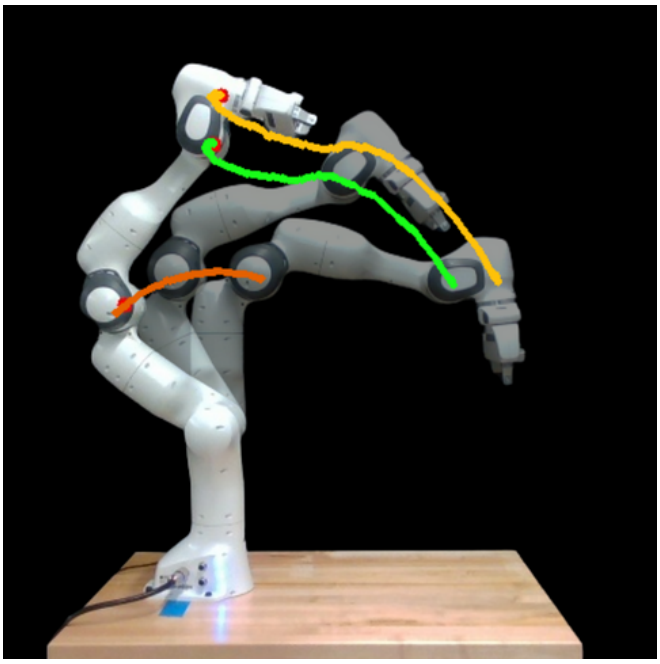

Fig. 1. The proposed keypoint-based adaptive visual servoing method is utilized to identify keypoints on the robot arm. The three trajectories follow the keypoints on the initial configuration, one intermediary configuration and converge to the goal configuration.

In our prior work, we presented the first visual servoing algorithm in the literature that can control the robot configuration purely based on visual information [8]. In that work, we extracted a skeleton model of the robot's shape from its depth images, fit a B-Spline curve to the skeleton, and used the control points of the B-Spline as control features of an adaptive IBVS algorithm. We obtained successful results in making the robot converge to desired configurations, and even achieved better transient response compared to controlling the robot only using end effector features. However, we also identified that skeleton representations were noisy (due to noisy depth images) and computationally heavy to extract. Also, when a gripper is attached to the robot, the skeleton shape deforms and branches at its tip introducing further noise to the control points. As a consequence of this, we could not get rid of markers altogether, and needed to use them for constraining the two ends of the extracted

*This paper was supported in part by the National Science Foundation under grant IIS-1900953.

[1]S Chatterjee, [1]A C Karade, [1]A Gandhi, and [1]B Calli are with Department of Robotics Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA schatterjee@wpi.edu, akarade@wpi.edu, agandhi2@wpi.edu

skeleton. Therefore, while we achieved promising results for configuration control of the robot using purely visual information, the mentioned factors motivated us to focus on identifying more robust natural features (keypoints).

Regarding robust keypoint detection, we were inspired by the success of human body pose estimation and tracking frameworks. Several of these algorithms detect the location of specific keypoints corresponding to body parts, e.g. hands, shoulders, eyes. There are ample datasets available for human body pose estimation (e.g. MPII Human Pose [9], COCO test-dev [10], Densepose-COCO [11]) as well as frameworks like OpenPose [12] and DeepPose [13] providing deep learning architectures for training pose detectors.

For detecting keypoints on robots, the literature is much sparse. [14] presents a large dataset of three robotic manipulators in simulation, and created a deep learning framework, DREAM, to detect keypoints on real robots. The main focus of that paper is one-shot camera calibration, i.e. the algorithm estimates the camera extrinsics via the detected keypoints on the robotic manipulator. While the algorithm delivers very good performance for its purpose, it is not tailored for continuous detection of keypoints, which is required by our control application. This is because the training is done purely on the simulated images of the robot, and when applied to a real setting, the algorithms cannot provide continuous keypoint detection reliably due to sim-to-real differences. Nevertheless, this work is a very important milestone for our framework, and was used as one of our building blocks. Similarly, a very recent work published in [15], where deep learning methods are used to detect keypoints on a robot using a simulation dataset.

This paper has the following novel contributions:

- We develop and publish an open source pipeline to collect a dataset with real robots for training keypoint detection algorithms. The pipeline does not require manual labeling. By allowing to easily train with real robots in their operating environments, we are able to achieve robust keypoint tracking performance that is suitable for control purposes. The prelude process of generating the dataset and training the network is a one time process unless the environment changes significantly. While we applied it to Franka Emika Panda type robot, our pipeline can be utilized to train keypoint detectors for other manipulators as well. Our code is available at `https://github.com/JaniC-WPI/KPDataGenerator.git`
- We have customized deep learning model Keypoint RCNN [16] to train the collected data to predict keypoints on a robot arm. In literature Keypoint RCNN has been used primarily for predicting keypoints on human body [17], [18].
- We utilize the keypoints in an adaptive IBVS controller [19] to achieve configuration space convergence via natural features. In this way, we completely removed the dependence on the end effector markers, and allow control with purely visual information without relying on robot Jacobians or joint encoder readings.
- We achieved robust planar (2D) control results and compared our method's performance to the skeleton-based visual servoing approach [8], which is the only other configuration space controller to the best of our knowledge. We experimentally showed that the keypoints are more robust and less noisy than the skeleton representations, and provide better transient response.
- We present proof of concept experiments for keypoint-based control of the robot in 3D. To the best of our knowledge, these are the first 3D results for controlling the robot configuration purely based on natural visual features.

II. Keypoint-Based Adaptive Visual Servoing

The overall pipeline can be described by the flowchart depicted in Fig. 2 and 3. It comprises of four parts: keypoint identification, data collection, network architecture and keypoint prediction, and adaptive visual servoing.

A. Keypoint Identification:

In this section, we present an approach for keypoint identification on a robot manipulator. We assume a pinhole camera model [20] and follow the terminology in [21]. As shown in Fig. 2, we use the camera intrinsics, camera extrinsics, and the forward kinematics of the arm to obtain 2D image projections of a particular point in the world frame using the following equation:

$$s\, p = A \left[R|t\right] P_w \tag{1}$$

where P_w is a 3D point represented with respect to the world coordinate system, p is the 2D project of that point on the camera's image plane, A is the camera intrinsics matrix (a matrix of camera parameters including focal length, optical center and lens distortion coefficients), R and t are the rotation and translation of the camera that describe the transformation from the world frame to the camera frame (camera extrinsics), and s is the projective transformation's arbitrary scaling . Our goal is to solve for p for joint frame locations of the robot, i.e. we find the projections of each joint frame origin on the image plane. From eq. (1) p can be solved as

$$p = sA(RP_w + t) \tag{2}$$

(keeping s the same since it is an arbitrary scalar).

In our implementation, the world frame in question is the base of the Franka-Emika arm. The camera intrinsics, that is A, is obtained from the topic published by the ROS Wrapper of Intel Realsense2 D435i camera. We obtain the camera extrinsics by using the DREAM framework [14]. The DREAM framework provides 'one shot' camera calibration, i.e. it outputs camera extrinsics using only a single RGB image of robots. This process needs to be run once, if/when camera viewpoint changes (while DREAM provides decent performance in general, it does not provide correct results every time. Therefore, in practice, we run it multiple times for different robot poses until we get consistent results for our camera extrinsics). We read the joint positions from

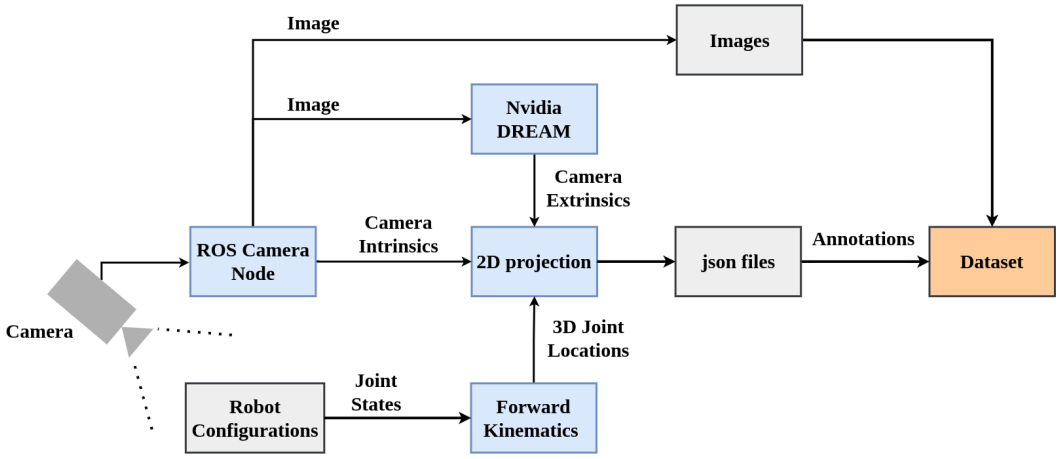

Fig. 2. Overall data collection pipeline

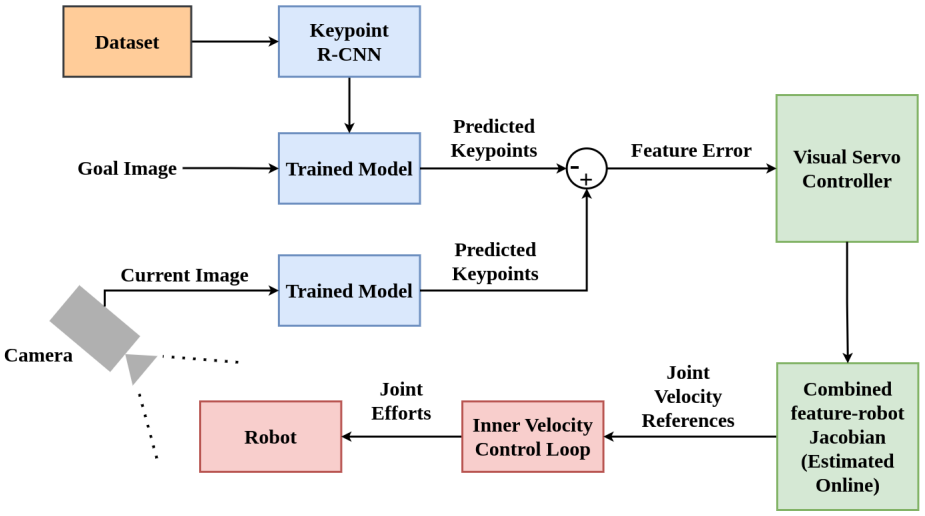

Fig. 3. Visual servoing applying predicted keypoints from Keypoint RCNN model

robot encoders and used them to calculate the forward kinematics [22] to obtain respective joint frame locations with respect to the world frame as depicted in Fig. 4. For identifying keypoints on the robot gripper as well, we applied an offset of $0.1070m$ in z axis to the frame origin of the seventh joint, and from there applying offsets of $0.09m$ and $-0.09m$ in x direction to obtain (*ee_1*, *ee_2*). We also applied another set of transforms to obtain keypoints corresponding on the gripper fingers as *panda_finger_1* and *panda_finger_2*. We then calculate the 2D projections of each joint frame locations using eq. (2). In this way we obtain 10 natural keypoints on the robot.

B. Data Collection:

We created an automated pipeline, which captures several images of the Franka Arm in arbitrary configurations and computed the keypoints location for each configuration as described in Section II-A (code is available at https://github.com/JaniC-WPI/KPDataGenerator.git). Along with the keypoints, we also computed a bounding box with each keypoint as the center. The keypoints and bounding boxes are then saved in a json file corresponding to each image frame. The 'keypoints' field in json file is of the format $[x, y, v]$, where v is the visibility of the keypoint and x and y are the pixel coordinates of the keypoint in image space. The 'bboxes'(bounding boxes) field have the format $[x-(bb\_size/2), y-(bb\_size/2), x+(bb\_size/2), y+(bb\_size/2)]$, where bb_size is the size of each side of the bounding box and x and y are the pixel co-ordinates of the keypoints as mentioned above. In our case, we set bb_size to 20. As mentioned in Section II-A, we have 10 keypoints and corresponding bounding boxes for our manipulator. Please refer Fig. 4 the corresponding image with labels on keypoints for the following sample json file, created from the data collected using the above pipeline. Here 'id' refers to the specific number of image in the dataset, 'image_rgb' is the

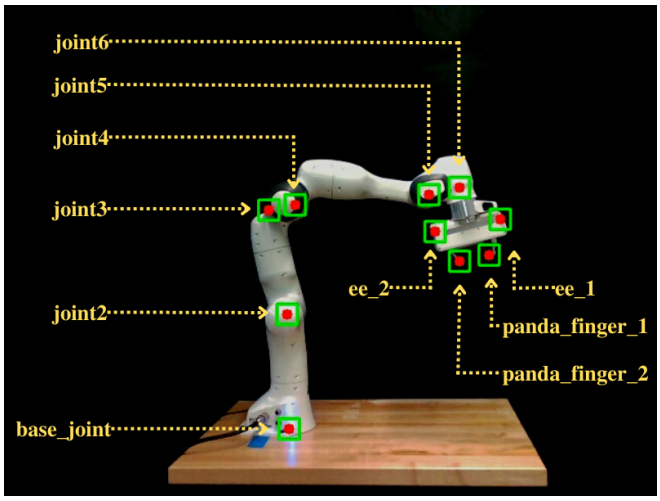

Fig. 4. Sample image from the dataset collection pipeline labeled with the keypoints, bounding boxes and respective joint names.

name of the corresponding image. 'bboxes' are the list of bounding boxes as computed above for each keypoint and 'keypoints' are the list of keypoints for each joint:


```
{
    "id": 897,
    "image_rgb": "000897.rgb.jpg",
    "bboxes": [
        [269.3287503494668, 404.37420417148,
             289.3287503494668, 424.37420417148],
        [266.6799207638364, 292.44542697801563,
             286.6799207638364, 312.44542697801563],
        [249.48124504518523, 190.33305712750948,
             269.48124504518523, 210.33305712750948],
        [274.7581009597627, 184.64644505138133,
             294.7581009597627, 204.64644505138133],
        [405.513941193429, 175.24921979029767,
             425.513941193429, 195.24921979029767],
        [436.33729934578105, 168.1752442805069,
             456.33729934578105, 188.1752442805069],
        [476.248986004589, 198.69040529666674,
             496.248986004589, 218.69040529666674],
        [412.1812597605679, 210.9391809017749,
             432.1812597605679, 230.9391809017749],
        [465.43534757619994, 234.48159405389336,
             485.43534757619994, 254.48159405389336],
        [436.4831059469089, 239.89902923224508,
             456.4831059469089, 259.8990292322451]
    ],
    "keypoints": [
        [[279.3287503494668, 414.37420417148, 1]],
        [[276.6799207638364, 302.44542697801563, 1]],
        [[259.48124504518523, 200.33305712750948, 1]],
        [[284.7581009597627, 194.64644505138133, 1]],
        [[415.513941193429, 185.24921979029767, 1]],
        [[446.33729934578105, 178.1752442805069, 1]],
        [[486.248986004589, 208.69040529666674, 1]],
        [[422.1812597605679, 220.9391809017749, 1]],
        [[475.43534757619994, 244.48159405389336, 1]],
        [[446.4831059469089, 249.89902923224508, 1]]
    ]
}
```


C. Network Architecture and Keypoint Prediction:

Pytorch's vision library [23] provides the *keypointrcnn_resnet50_fpn* [16] deep learning model, which is pre-trained to detect 17 keypoints in human body. We modified this model for the dataset that we created with 10 keypoints. This model used the backbone of Resnet50 with pre-trained weights of IMAGENET1K_V2 from COCO Dataset [10]. We created a custom Pytorch Dataset class, where we labeled each keypoint/bounding boxes from 1-10 (*base_joint* being 1 and *panda_finger_2* being 10). We fine-tuned the model on our dataset using Stochastic Gradient Descent with a learning rate of 0.001 and a momentum of 0.9. We also applied a weight decay of 0.0005 to avoid overfitting. We performed data augmentation on the training set to diversify the data. We added one image per batch and ran the training for 25 epochs. We also customized the evaluation metrics provided by pycocotools library of cocoapi [10], to evaluate the model using mean Average Precision(mAP). Table I shows the mAP for the model we used for our inference. We predicted the keypoints using the trained model by checking the score for each label. A sample label to score output looks like the following:

'labels': tensor([2, 1, 3, 4, 5, 9, 8, 10, 6, 7, 8, 5, 2, 6, 3, 1, 7, 9, 4, 10], device='cuda:0'), 'scores': tensor([1.0000, 0.9999, 0.9996, 0.9994, 0.9988, 0.9979, 0.9977, 0.9964, 0.9959, 0.9924, 0.0918, 0.0913, 0.0900, 0.0896, 0.0894, 0.0892, 0.0892, 0.0891, 0.0891, 0.0888], device='cuda:0'). In the 'labels' tensor, the joints are ranked as per their score in the 'scores' tensor. As we can see in the 'labels' tensor, there are repetitions.

As depicted above there are repetitions of labels. We first select the labels/keypoints with score > 0.7. We then apply Non-Maximum Suppression(NMS) procedure to select the best keypoints. NMS method keeps only the labels with an IoU(Intersection Over Union) $= 0.3$.

TABLE I
KEYPOINTS EVALUATION

Average Precision(AP)	
AP at IoU=.50:.05:.95	0.607
AP at IoU=.50	0.788
AP at IoU=.75	0.632
Average Recall(AR)	
AR at IoU=.50:.05:.95	0.696
AR at IoU=.50	0.849
AR at IoU=.75	0.718

D. Adaptive Visual Servoing

The keypoints inferred from our trained model, as described in Section II-C, are used as image features in an adaptive visual servoing scheme. We define a feature error vector $\mathbf{e}$ between the current keypoint locations p and the desired keypoint locations p^* as shown in (3).

$$\mathbf{e} = p - p^* \tag{3}$$

We estimate a combined eye-hand-Jacobian $\hat{J}_c$, that relays the change in feature locations in image space ($\dot{p}$) to the robot's joint velocities ($\dot{q}$), as shown in our prior work [8].

$$\dot{Q} = [\dot{q}[k-n+1],\ \dot{q}[k-n+2],\ \ldots,\ \dot{q}[k]]^T \tag{4}$$

$$\dot{P} = [\dot{p}[k-n+1],\ \dot{p}[k-n+2],\ \ldots,\ \dot{p}[k]]^T \tag{5}$$

Specifically, we collect pairs of visuo-motor data for the last n samples of applied robot joint velocities $\dot{Q}$, as shown in (4) and change in image feature (keypoint) locations $\dot{P}$, as shown in (5). The collected window of visuo-motor data pairs is used in a single-shot least squares optimization to obtain adaptive updates, for each row of $\hat{J}_c$, as described in (6). The adaptive updates are added to the current estimation of $\hat{J}_c$ as shown in (7).

$$\Delta \hat{J}_{c_i}[k] = \gamma \dot{Q}^T (\dot{Q} J_{c_i}[k] - \dot{P}) \tag{6}$$

$$\hat{J}_{c_i}[k+1] = \Delta \hat{J}_{c_i}[k] + \hat{J}_{c_i}[k] \tag{7}$$

Finally, the image feature error is utilized with a proportional control law and the current estimate of $\hat{J}_c$, as shown in (8), to generate reference joint velocities that drive the robot to its desired configuration in the image.

$$\dot{\mathbf{q}}_\mathbf{r} = -\lambda J_c^+ \mathbf{e} \tag{8}$$

In our implementation, we initialize the adaptive scheme online by providing the system with a small excitation velocity trajectory to estimate an initial combined Jacobian. After this initial estimation, the algorithm continuously updates the Jacobian in runtime during control. For adaptive schemes, it is important to select the window size carefully. Large window sizes prevent sudden spikes in robot joint velocities but increase the convergence time of the estimated combined Jacobian whereas smaller window sizes may cause sudden changes and spikes in the joint velocities generated from the estimated Jacobian.

III. Experiments and Results:

In this section, we analyze the performance of the keypoints and their capabilities as control features in the adaptive visual servoing scheme. We also compare it with the skeleton-based approach [8], which is the only other algorithm for purely vision-based configuration space visual servoing.

A. Feature Robustness:

The keypoints from the proposed method have only negligible noise. We compare the b-spline control points that are obtained from the robot skeleton in [8] to our keypoint features. We run both algorithms for detecting features when the robot is moving on the same trajectories. For these experiments, for our keypoint-based algorithm, we only used keypoints located at *joint4*, *joint5* and *joint6* to have a more clear comparison, since the skeleton also has 3 control points at similar locations. As seen in Fig. 5, the keypoints provide much more robust and less noisy features. Also, it is important to note that the skeleton-based algorithm in [8] uses ArUco markers (also shown in Fig. 5) to simplify the image processing steps and to obtain more consistent skeletons. Our keypoint features provide superior performance without requiring any markers. (Here we put a side note that the reader may have noticed that the b-spline control points of the skeleton-based method are not on the robot; this is a normal outcome, since control points of b-splines does not fall on the curve that they are fit to. This particular phenomenon does not constitute a problem for vision-based control purposes. Here, we would like to bring the attention to the noise level difference between the two algorithms.)

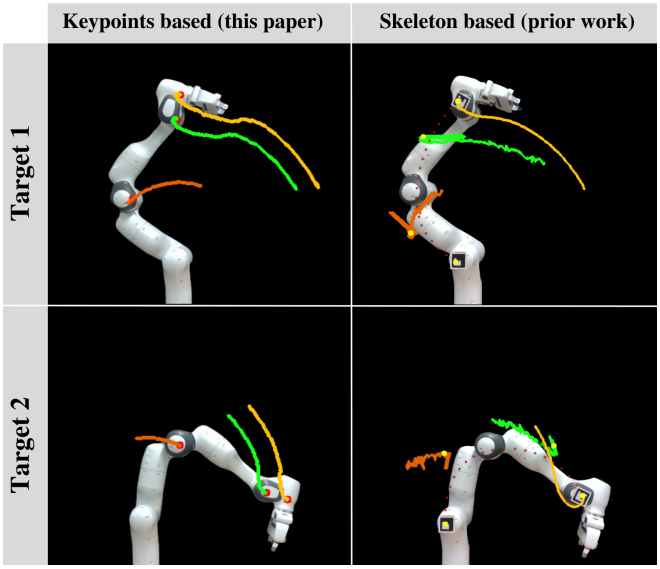

Fig. 5. Trajectory comparison between proposed method's keypoints and existing method's control points. The trajectory of the end effector of skeleton method looks cleaner because the control point is marked by a fiducial marker an the end effector location.

TABLE II
Performance Comparison Summary

	Skeleton Based	Proposed algorithm
System Rise time (s)	7.90 ± 2.70	6.06 ± 2.16
System Settling time (s)	13.90 ± 6.90	11.10 ± 2.47
End effector Rise time (s)	7.30 ± 2.52	5.22 ± 1.63
End effector Settling time (s)	13.20 ± 6.50	9.75 ± 3.72
Overshoot (%)	4.22 ± 6.41	2.70 ± 2.79

B. Transient Response Performance:

We ran experiments using the Franka Emika Panda Arm to analyze the performance of the keypoint-based adaptive visual servoing algorithm and to compare it to the skeleton-based approach in [8]. We used Intel RealSense D435i camera to acquire the visual feedback. These experiments are restricted to planar case as the skeleton-based approach so far has only worked for the planar case. We tuned the algorithm parameters of both methods for best performances. The estimation window size for the adaptive visual servoing algorithm (n in eqs. (4) and (5)) is set to 50 for skeleton based approach, but 20 for the proposed method. We believe, the window size can be kept small in the proposed method because of the lack of noise in the features. A control loop

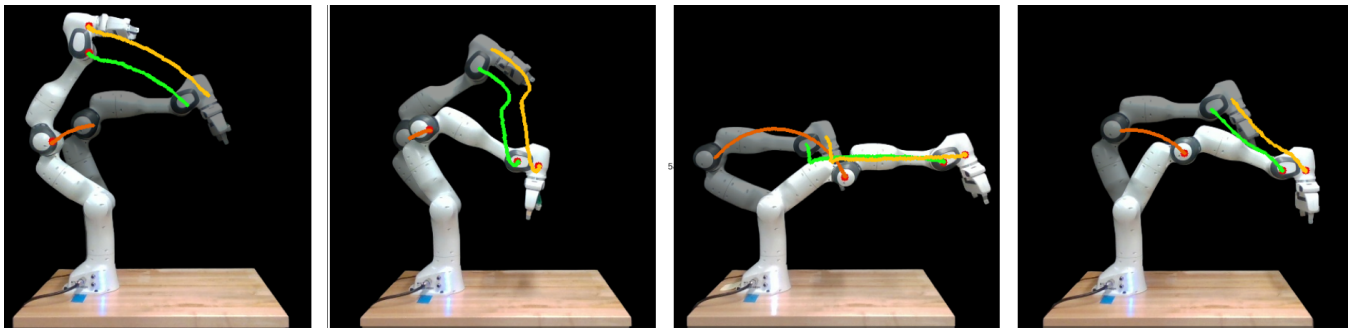

Fig. 6. Feature trajectory plots for different robot configurations for our proposed algorithm. The robot servos from the slightly transparent initial configurations to the solid ones.

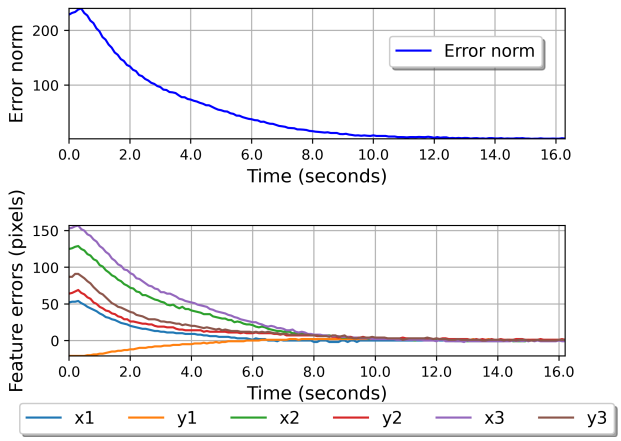

Fig. 7. Image feature error norm (top) and individual image feature errors (below).

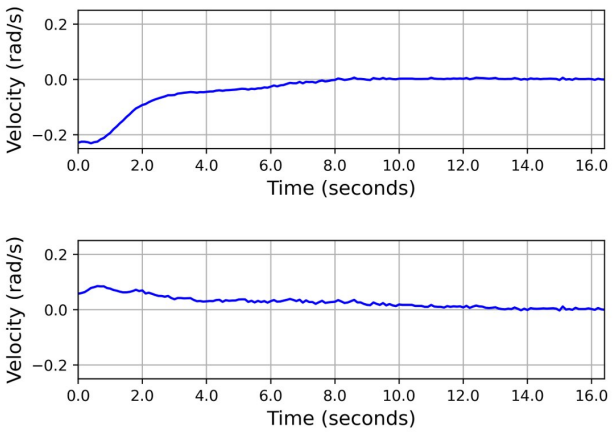

Fig. 8. Applied joint velocities. Joint-1 (top) and joint-2 (bottom).

frequency of $10Hz$ is used. Both algorithms are provided with the same velocity profiles for collecting the first n samples required for the adaptive algorithm to get the very intial Jacobian estiamtes. The controller gains, λ in eq. (8), and adaptation gain γ in eq. (6) are tuned extensively so that both the methods perform their best: we experimentally tuned the gains to get the lowest rise time while keeping the overshoot at the end effector position less than 5%.

We conducted 18 experiments for each method. Reference locations for image features are obtained by taking images of the robot at the desired locations. The mean and standard deviation of rise time, settling time, overshoot for each algorithm is given in Table II. The maximum rise time and settling time among the three features are the system rise time and settling time. The end effector rise time and settling time are those of the features closest to the end effector. This data shows that our algorithm provides faster convergence with less overshoot. We believe the better transient response is due to the lack of noise in the data. Fig. 6 shows the trajectories of each features for the proposed method for different robot poses, from initial to target configurations. Fig. 7 depicts the normalized feature error as well as individual feature errors for the first experiment displayed. The joint velocities are depicted in Fig. 8 and the model error of the combined Jacobian is shown in Fig. 10. The plots represent the servoing phase. As the robot reaches close to convergence we stop updating the interaction matrix. The figures show how the method provides smooth decay of the error with near optimum trajectory even when the Jacobian error does not converge to zero.

TABLE III
SUMMARY OF REPEATABILITY TESTS

	Proposed algorithm
System Rise time (s)	5.820 ± 0.380
System Settling time (s)	10.680 ± 0.650
End effector Rise time (s)	4.320 ± 0.400
End effector Settling time (s)	10.520 ± 0.470
Overshoot (%)	1.042 ± 0.004

C. *Repeatability:*

We have taken a repeatability test for the proposed algorithm by running an experiment with same initial and reference position for 5 times. The results are summarized in Table III. As can be seen from these results, there is minimal variance in rise time, setting time and overshoot values between the 5 experiments.

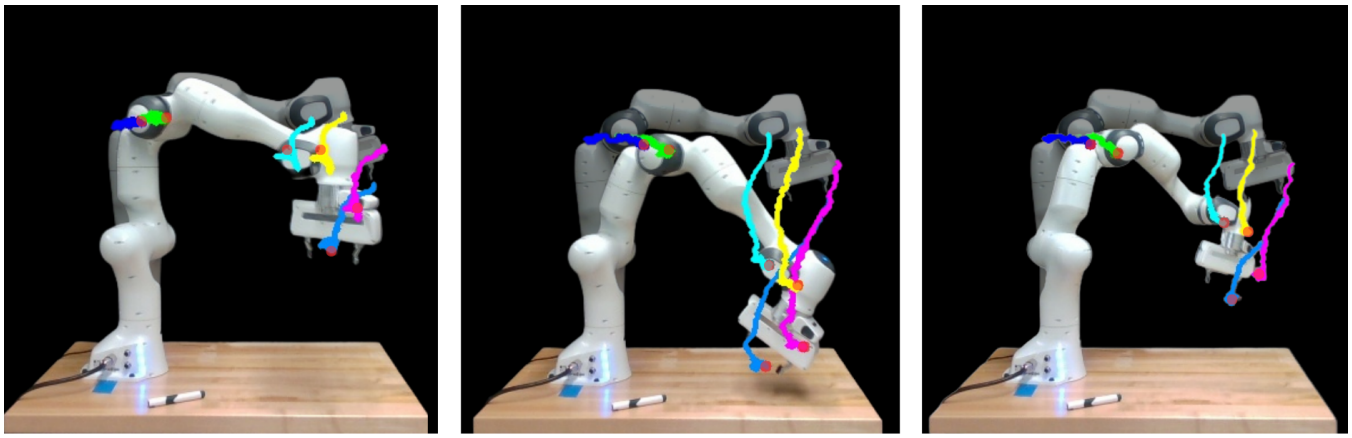

Fig. 9. Motion in 3D cartesian Space. The robot servos from the slightly transparent initial configurations to the solid ones.

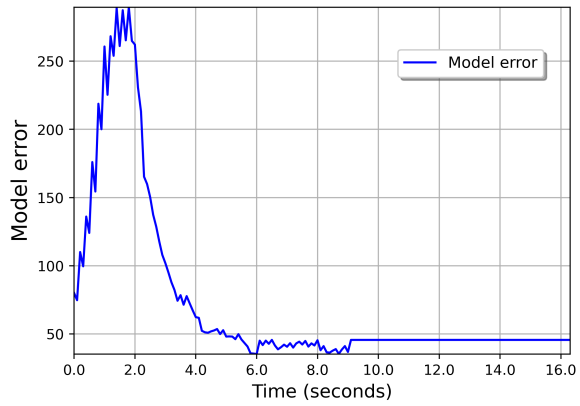

Fig. 10. Model error for estimating the combined Jacobian.

D. Motion in 3D Space:

We used the keypoint-based adaptive visual servoing algorithm to control the arm in 3D cartesian space, and obtained successful results. Fig. 9 depicts three configurations of 3D motions from the same initial position. To the best of our knowledge, these are the first 3D results in the literature that utilize natural keypoints in a purely vision-based configuration control framework (please note that we don't even use joint position measurements or any apriori Jacobian information for the performance in the runtime). However, we observed two limitations. 1) The robot was not able to converge to the 3D references that are too much away from the 2D plane that we used for collecting most of our data. We believe this is due to not having enough training data for out of place configurations of the robot. This is also evident from the fact that the features on the robot moves away from their expected locations, as the robot goes more away from the 2D plane. However, the detected keypoint locations for each robot configuration are often consistent, leading to successful results for the vision-based configuration control. We believe that this feature accuracy issue can be solved by collecting more out-of-plane data using our data collection pipeline and training a new keypoint detection model. 2) While we used the same set of gains for all the mentioned 2D experiments above, we had to tune the control gains individually for each experiment shown in Fig. 9. As a future work, we will investigate methods to improve the control aspects for the 3D motions in the configuration space.

IV. Conclusion and Future Work:

In this paper, we were able to control the robot in its planar configuration space using keypoints inferred from a modified Keypoint RCNN deep learning model. We first identify keypoints on the robot's body using the robot's kinematics and the camera parameters. We then create a data collection pipeline to collect images of different robot configurations with corresponding keypoints annotations. We create a dataset using the pipeline and train a model using this dataset, by modifying the Keypoint RCNN model. The saved model is used to predict keypoints on the manipulator in real time. These predicted keypoints are used as visual features to control the manipulator using an adaptive visual servoing scheme. We compared the performance of our proposed method with the existing method of controlling a manipulator in configuration space. We observed the proposed keypoints are much robust and gives better control performance in an average. Controlling the arm using keypoints is also repeatable. We had shown proof of concept experiments for 3D cartesian space control. In the future, we look forward to improve the 3D motion control performance. This also includes expanding on our data set, and possibly some changes in our network architecture to more robustly acquire the keypoints during 3D motions. We also plan to utilize this vision-based control framework for executing manipulation task, i.e. using the finger joint keypoints, we plan to execute pick-and-place tasks via visual servoing. We are also looking forward to apply keypoints based control in soft or deformable robots.

REFERENCES

[1] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 651–670, 1996.

[2] F. Chaumette, S. Hutchinson, and P. Corke, *Springer Handbook Of Robotics*. Springer, 2016, vol. Springer.

[3] M. Jagersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 4, pp. 2874–2880, 1997.

[4] J. Lai, K. Huang, B. Lu, and H. K. Chu, "Toward vision-based adaptive configuring of a bidirectional two-segment soft continuum manipulator," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, vol. 2020-July, pp. 934–939, 2020.

[5] B. Calli and A. M. Dollar, "Vision-based precision manipulation with underactuated hands: Simple and effective solutions for dexterity," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem. IEEE, oct 2016, pp. 1012–1018.

[6] H. Cuevas-Velasquez, N. Li, R. Tylecek, M. Saval-Calvo, and R. B. Fisher, "Hybrid multi-camera visual servoing to moving target," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1132–1137.

[7] P. Ardón, M. Dragone, and M. S. Erden, "Reaching and grasping of objects by humanoid robots through visual servoing," in *Haptics: Science, Technology, and Applications: 11th International Conference, EuroHaptics 2018, Pisa, Italy, June 13-16, 2018, Proceedings, Part II 11*. Springer, 2018, pp. 353–365.

[8] A. Gandhi, S. Chatterjee, and B. Calli, "Skeleton-based adaptive visual servoing for control of robotic manipulators in configuration space," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2022, pp. 2182–2189.

[9] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, 2014, pp. 3686–3693.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[11] R. A. Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7297–7306.

[12] Z. Cao, G. Hidalgo, T. Simon, S. E. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, pp. 172–186, 1 2021.

[13] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660.

[14] T. E. Lee, J. Tremblay, T. To, J. Cheng, T. Mosier, O. Kroemer, D. Fox, and S. Birchfield, "Camera-to-robot pose estimation from a single image," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9426–9432.

[15] J. Lu, F. Richter, and M. C. Yip, "Pose estimation for robot manipulators via keypoint optimization and sim-to-real transfer," *IEEE Robotics and Automation Letters*, vol. 7, pp. 4622–4629, 4 2022.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[17] H. Wang, L. Huang, K. Yu, T. Song, F. Yuan, H. Yang, and H. Zhang, "Camper's plane localization and head pose estimation based on multi-view rgbd sensors," *IEEE Access*, vol. 10, pp. 131 722–131 734, 2022.

[18] X. Deng, J. Liu, H. Gong, H. Gong, and J. Huang, "A human-robot collaboration method using a pose estimation network for robot learning of assembly manipulation trajectories from demonstration videos," *IEEE Transactions on Industrial Informatics*, 2022.

[19] A. Shademan, A.-M. Farahmand, and M. Jägersand, "Robust jacobian estimation for uncalibrated visual servoing," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 5564–5569.

[20] Camera calibration and 3d reconstruction. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[21] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, 11 2000.

[22] J. J. Craig, *Introduction to Robotics Mechanics and Control*, 3rd ed. Pearson Education International, 2005.

[23] V. Ayyadevara and Y. Reddy, *Modern Computer Vision with PyTorch: Explore deep learning concepts and implement over 50 real-world image applications*. Packt Publishing, 2020.