Review of Estimating 3D Motion and Forces of Person-Object Interactions from Monocular Video

Balthazar Neveu
ENS Paris-Saclay
Saclay, France
balthazar.neveu@ens-paris-saclay.fr

Matthieu Dinot
Ecole Polytechnique
Palaiseau, France
matthieu.dinot@polytechnique.edu

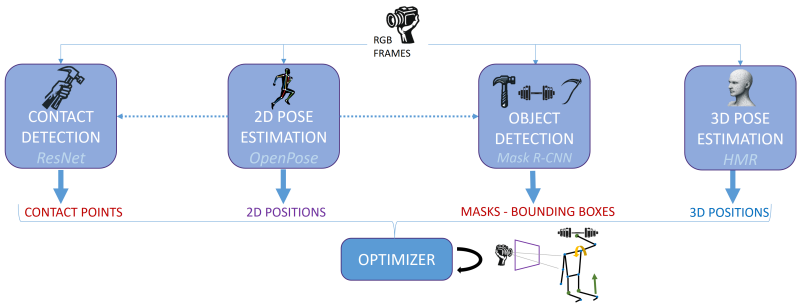

Figure 1: Full original pipeline. A multiple stage vision pipeline extracts poses and contact data of the human and the object. Full body dynamics (joint angles, torques and external forces) are then reconstructed during the optimization stage.

KEYWORDS

Perception and Estimation, Optimal Control, Inverse Problems

1 INTRODUCTION

Humans can easily learn how to perform a given movement or manipulate an object by observing others and attempting to replicate their actions. In robotics, this behavioral cloning ability would be usefull to train humanoid robots to perform such tasks. However, unlike humans, computers still struggle to understand complex human-object interactions from visual data alone. Traditional methods like motion capture offer a solution, but they come with high costs and do not leverage the abundant instructional video resources available online.

To address this challenge, Li et al. [7] proposed a novel approach to estimate 3D motion and forces of human-object interactions from single RGB videos.

In this work, we critically reviewed and reimplemented certain aspects of their method. Our initial goal was to assess the reproducibility and robustness of the original findings, as well as to explore potential enhancements. We focused on understanding the underlying methodology, adapting it to a simplified pipeline. Our code is available on GitHub.

2 METHODOLOGY OF THE ORIGINAL PAPER

The study by Li et al. [7] aimed to reconstruct the 3D motion of a person interacting with an object, including the position and applied forces, from a single RGB video. Their methodology unfolds in two principal steps. The first step involves the application of computer vision techniques to accurately determine the human and object poses. Vision pipeline is also in charge of detecting whether or not certain specific joints are in contact with the object or the ground. The second step solves an optimization problem, rooted in control theory principles, to fully recover the dynamic motion. The full pipeline is depicted in fig. 1.

2.1 Retrieve Human and Object 3D Pose and Contact from RGB Video

Human 2D Pose Estimation. For the task of human pose estimation, the authors used a pretrained version of OpenPose [2]. This tool allowed them to identify and track human joint positions in two dimensions from the RGB video.

Contact Recognition. The recognition of contact points, being whether specific joints (like feet, hands, and neck) were in contact with the ground or the object, was achieved by first cropping areas in the video frame around the joints identified by OpenPose. These cropped images were then processed through the corresponding joint's ResNet [5], which had been fine tuned on a custom dataset to discern contact.

Object 2D Endpoint Detection. In order to estimate the 2D positions of object endpoints, the authors relied on instance segmentation capabilities of Mask R-CNN [4]. This method was applied to different classes of objects (such as barbells, hammers, scythes, and spades) identified in their video dataset. Training Mask R-CNN separately for each object class, enabled the extraction of segmentation masks and bounding boxes from the video frames. These were then used to estimate the 2D locations of the object's endpoints.

Human 3D Pose Estimation. The authors also used HMR (Human Mesh Recovery) [6] for 3D human pose estimation, although this is not clearly stated in their method overview.

2.2 Reconstruct Person-Object Dynamic Motion

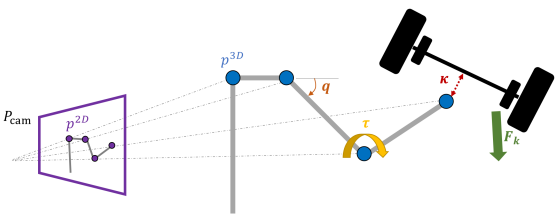

Figure 2: Retrieving full body dynamics (muscle torques τ and external forces F_k from 2D and 3D pose observations extracted from the video. Configuration state q of the body determines the 3D joints positions $p^{\mathbf{3D}}$ which are projected as $p^{\mathbf{2D}}$ on the pinhole camera sensor plane. The red dashed arrow illustrates that the distance between the object and the hand shall be minimized to statisfy the contact constraint κ.

The optimization framework presented by Li et al. [7] is designed to estimate the 3D trajectories of both the human (h) and the object (o), as well as the applied contact forces, by solving the following trajectory optimization problem:

$$\begin{aligned} \underset{x,u,c}{\text{minimize}} \quad & \int_0^T l^h(x,u,c) + l^o(x,u,c)dt, \\ \text{subject to} \quad & \kappa(x,c) = 0 \quad \text{(contact motion model)}, \\ & \dot{x} = f(x,c,u) \quad \text{(full-body dynamics)}, \\ & u \in \mathcal{U} \quad \text{(force model)}. \end{aligned}$$

Variables. The state variables $x \coloneqq (q^h, q^o, \dot{q}^h, \dot{q}^o)$ encompass the configuration and velocities of the human and object, while the control variables $u \coloneqq (\tau_m^h, f_k, k = 1, \ldots, K)$ include the muscle torques and contact forces at the K contact points. The contact state c represents the presence and locations of contacts throughout the video sequence.

Although not explicitly stated in the paper, it should be noted that in their implementation, the velocities ($\dot{q}^h$ and $\dot{q}^o$) are not direct optimization variables. Instead, velocities are approximated using the backward difference scheme $\dot{q}_t = (q_t - q_{t-1})/\Delta t$.

Loss Functions. The loss functions l^h for the human consists of the following components:

- $l_{2D} = \sum_j \rho\left(p_j^{2D} - P_{\text{cam}}\left(p_j(q)\right)\right)$: A term ensuring 2D consistency by minimizing the error between the observed 2D joint positions p_j^{2D} given by OpenPose and their reprojections from the 3D estimates $p_j(q)$, using the camera projection matrix P_{cam}. Outlier influence is mitigated by employing the Huber loss function ρ.
- $l_{3D} = \sum_j \rho\left(p_j^{3D} - p_j(q)\right)$: This loss ensures that the estimated 3D positions $p_j(q)$ adhere closely to the HMR 3D references p_j^{3D}.
- $l^h_{\text{pose}} = -\log\left(p\left(q; \text{GMM}\right)\right)$: A likelihood term for the human pose, encouraging plausible human poses through a Gaussian Mixture Model (GMM) trained on motion capture data.
- $l^h_{\text{torque}} = \|\tau_m^h\|^2$: A regularization term penalizing high muscle torque values to favor energy-efficient movements.
- $l_{\text{smooth}} = \sum_j \|v_j(q,\dot{q})\|^2 + \|\alpha_j(q,\dot{q},\ddot{q})\|^2$: Term penalizing fast 3D linear velocities $v_j(q,\dot{q})$ and linear accelerations $\alpha_j(q,\dot{q},\ddot{q})$. This term encourages smooth joint trajectories.
- Force-related terms: These encompass additional forces and contact forces that were not the focus of further investigation in our study.

The loss function for the object, l^o, is constructed similarly, omitting the 3D consistency, torque smoothness and pose likelihood terms.

Constraints. The optimization process is subject to constraints that govern the full-body dynamics of the human and object interaction. These dynamics are encapsulated by the Lagrange dynamics equation:

$$M(q)\ddot{q} + b(q,\dot{q}) = g(q) + \tau,$$

where $M(q)$ is the generalized mass matrix, $b(q,\dot{q})$ represents the centrifugal and Coriolis forces, $g(q)$ is the generalized gravity vector, and τ denotes the joint torque contributions. Here, $\dot{q}$ and $\ddot{q}$ are the joint velocities and accelerations, respectively. This constraint ensures that the estimated motion adheres to the fundamental principles of physics governing movement.

Other constraints, such as those related to contact, are not the focus of our study and thus are not elaborated upon here.

Implementation. The authors approached the problem's solution by discretizing it and applying backward difference scheme for derivatives, specifically $\dot{y}_t = (y_t - y_{t-1})/\Delta t$. The computation of kinematic and dynamic quantities was facilitated by the Pinocchio library [3]. Optimization was performed using the Levenberg-Marquardt algorithm, within the Ceres Solver [1]. Due to the limitations on the constraint's type in such solvers, the constraints were incorporated as penalty terms in the objective function.

3 REIMPLEMENTATION STRATEGY

The code accompanying the orignial paper is rather difficult to use due to its inherent complexity and lack of documentation. To effectively explore the core contributions of the paper, we decided to reimplement parts of it from scratch with application to a simplified use case.

In our approach, we focused on reconstructing arm motion from video data, a simplification that allowed us to tackle the fundamental aspects of the methodology without the computational burden of full-body modeling.

Our code is available and documented on GitHub.

3.1 Overview

Our simplifications (depicted in fig. 3) regarding the original paper are listed below:

- Only the arm motion (shoulder, elbow and wrist) is considered and not the full body.
  - We end up with the arm state q being a quaternion describing the upper arm state and a single elbow angle for the forearm state.
  - Shoulder is fixed to the world frame.
- Do not deal with objects, ground and external contact forces.

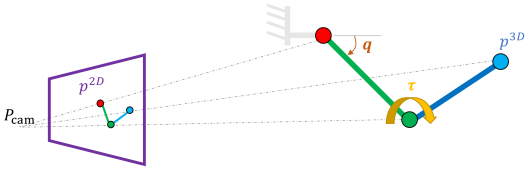

Figure 3: Simplified problem overview.

- Act under controlled video capture conditions.
  - A correct lighting and a static background to simplify the vision task.
  - We make sure that the camera is on a tripod and geometrically calibrated.

Regarding implementation, we started from scratch and wrote Python code without focusing on execution time. Instead of trying to re-implement the whole inverse dynamics optimization framework at once, we progressively built a solution. Since the arm has only 4 degrees of freedom, leading to 3x 3D joints, fitting the full camera parameters (extrinsics and intrinsics) at the same time as fitting the whole arm dynamics is not possible (under determined system).

- (3.2) - Vision pipeline extracts 3x 2D and 3x 3D arm estimated joints positions.
- (3.4) - We use inverse kinematics on the 3D points to get coarse robot arm state.
- (3.5) - We fit the camera/shoulder translation by minimizing the 2D reprojection error of the 3D points.

Due to lack of time, we were not able to implement the inverse kinematics which minimizes jointly the camera pose and the arm state q.

Finally, using the above q state estimation as an initialization, we implemented a proof of concept for the inverse dynamics optimizer (3.6) focusing on:

- minimizing l_{3D} the 3D reprojection error of the arm joints.
- under relaxed dynamic constraint.
- with smooth motion priors (velocities and accelerations) and minimal torque.

3.2 Vision Pipeline

For 2D and 3D human pose estimation, we used the MediaPipe framework [8] which performs the two tasks at once at inference time. It offers a more modern alternative to our updated version of Pytorch OpenPose. MediaPipe simplifies the vision pipeline by also replacing HMR 3D pose estimation in our usecase. An illustration of MediaPipe's inference results can be seen in fig. 4. We then only used the arm joint 2D and 3D positions.

Although we experimented with the contact recognizer component, we did not use it in our pipeline since we do not model contact.

3.3 Arm Model

We modelled the arm in Pinocchio [3]. The shoulder is represented by a spherical joint, allowing for three degrees of freedom. This

Figure 4: Shoulder, elbow, wrist joints indicated with dots are linked by the upper arm and forearm . 2D and 3D data is extracted from the full body pose estimated with MediaPipe.

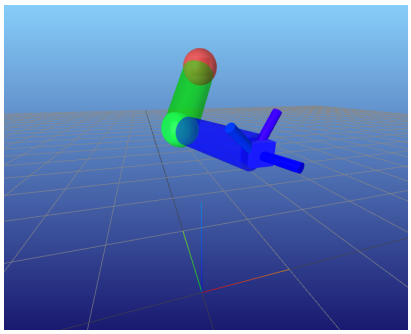

Figure 5: Arm Model visualisation

is expressed as a quaternion in the state vector q. The elbow is represented by a revolute joint, which permits rotation within a plane, resulting in a single degree of freedom. The state q therefore consists of five elements: the first four correspond to the shoulder quaternion, and the fifth represents the elbow angle, which is constrained within the range $[0, \pi]$ radians. The dimensions of the arm are fixed with the forearm and upper arm lengths fixed at 27 cm and 23 cm, respectively. See fig. 5

3.4 Inverse Kinematics

To get proper initializations to feed to the inverse dynamics optimizer, we relied on the 3D positions of the arm joints estimated by MediaPipe [8]. From a sequence of 3D positions of the arm joints, we used the inverse kinematics approach to estimate the joint angles. Inverse kinematics is highly relying on the Pinocchio library [3] to compute the Jacobian and perform forward kinematics (compute the 3D positions $M(t)_j$ of the joints from the joint angles $q(t)$).

Devil is in the details, and we had to be careful about the following points while implementing:

- We provide 2 tasks to the inverse kinematics solver: one for the elbow 3D position and one for the wrist 3D position.
- The mediapipe 3D estimations are not perfect but the most nottable issue is that the upper arm and the forearm lengths

are not constant across time. To overcome this limitation, 3D positions are re-scaled to match the expected nominal model arm length. This arm length normalization ensures that a solution exists but on the other hand, it may introduce reprojection errors.

Although nothing grants temporal smoothness of the estimated joint angles, we recursively feed the previous solution as an initial guess to the current timestep. This simple trick gives good results in practice (assessed qualitatively by the match between the reconstructed moving arm and the video).

3.5 Camera position

Problem formulation. Since the upper arm can rotate in all directions and since we're not using any priors to constrain the arm motion (at the shoulder level), moving the camera around the arm is equivalent to spinning the arm around itself. Our simplification unfortunately leads to a degenerate problem where the camera rotation may end up being redundant with the arm rotation. To avoid this issue, we decided to fix the camera orientation. We estimate the camera 3D position $T_{\text{cam}}(t)$ by jointly minimizing the 2D reprojection error of joints.

$$\min_{T_{\text{cam}}(t)} \sum_{j \in \text{shoulder, elbow, wrist}} \| K \cdot [Q_{\text{cam}} | T_{\text{cam}}(t)] \, \vec{P_{3D}}^{(j)}(t) - \vec{p_{2D}}^{(j)}(t) \|^2$$

where

- K is the camera 3x3 intrinsic matrix
- Q_{cam} is the camera orientation set to identity here I_3
- $T_{\text{cam}}(t)$ is the 3D camera position in the world frame (meters)
- $\vec{P_{3D}}^{(j)}$ are the homogeneous 3D coordinates of joint j (meters)
- $\vec{p_{2D}}^{(j)}$ are the homogeneous 2D coordinates of joint j in the sensor frame (pixels).

Camera calibration. To reduce the number of degrees of freedom, we always used the same camera through all our tests and we pre-calibrated the camera intrinsics K using Zhang's method [9]. Please refer to appendix A.1 for more details.

Camera pose optimization. Each timestep can be viewed as a separate optimization problem. We can also ensure better temporal coherence by adding a regularization term on the camera 3D position to the previous cost function $|| \frac{\Delta T_{\text{cam}}}{\Delta t} ||^2$. We perform a first quick pass using small windows of 2 second (60 frames) to initialize the camera position sequence (used for the poster session demo). In a second pass, we optimize over the whole sequence at once. Results are shown in fig. 6. A rule of thumb on a i7 - 8 core CPU is that this optimizer goes at 30fps (takes roughly 1 minute to optimize the camera pose a 1 minute long video).

3.6 Inverse dynamics optimizer

Given the simplifications discussed in section 3.3, we reformulated the optimization problem as follows:

$$\begin{aligned} &\underset{q, \tau_m}{\text{minimize}} && \int_0^T l(q, \tau_m)\, dt \\ &\text{subject to} && \dot{q} = f(q, \tau_m) \quad \text{(full-arm dynamics).} \end{aligned}$$

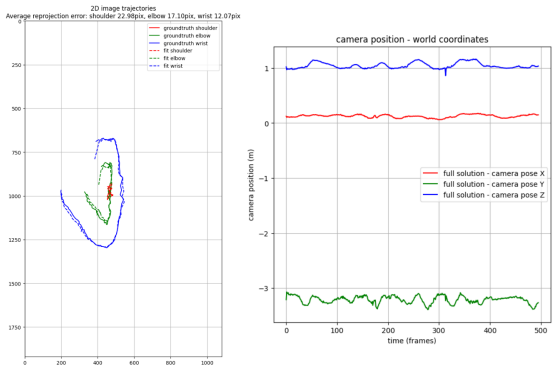

Figure 6: Left: projected arm 3D points onto the sensor after camera pose fitting - on the first 100 frames of the video. Average reprojection error reported here are computed on the 3 arm joints over the whole video. Right: Estimated distance from the camera to the shoulder is 3 meters wich is faithful to the real distance.

Variables. The model variables consist of the arm's configuration vector q and the muscle joint torques τ_m. We adopted the central difference scheme to approximate $\dot{q}$ for its enhanced precision and because it provides velocity information precisely at instant t.

$$\dot{q}_t = (q_{t+1} - q_{t-1}) / 2\Delta t$$

Acceleration was computed in a similar fashion.

Loss Functions. Our loss function l omits l_{2D} due to the already accurate 3D pose estimates and excludes the pose likelihood term since we target only the arm. The remaining terms include:

- l_{3D}: Ensures 3D position estimates align with the MediaPipe references.
- l_{torque}: A regularization term penalizing high muscle torque values to favor energy-efficient movements.
- l_{smooth}: Motion smoothness. This cost term penalizes rapid movements and accelerations.

Theses terms are the same than the corresponding ones in section 2.2.

Constraints. The arm dynamics are governed by the Lagrange dynamics equation, relaxed as an additional cost term.

$$\| \underbrace{M(q)\ddot{q} + b(q, \dot{q}) - g(q)}_{\text{Estimated torque from states } q} - \tau \|$$

The computation of this term heavily relies on RNEA (Recursive Newton Euler algorithm) implemented in Pinocchio [3].

Implementation. We discretized the continuous-time problem, and use the central difference scheme for derivative approximation. Pinocchio facilitated the computation of kinematic and dynamic quantities. The optimization was carried out using the Levenberg-Marquardt algorithm implemented in the SciPy library, chosen for its ease of integration with our Python-based pipeline. Constraint enforcement was achieved by embedding them as penalty terms within the loss function.

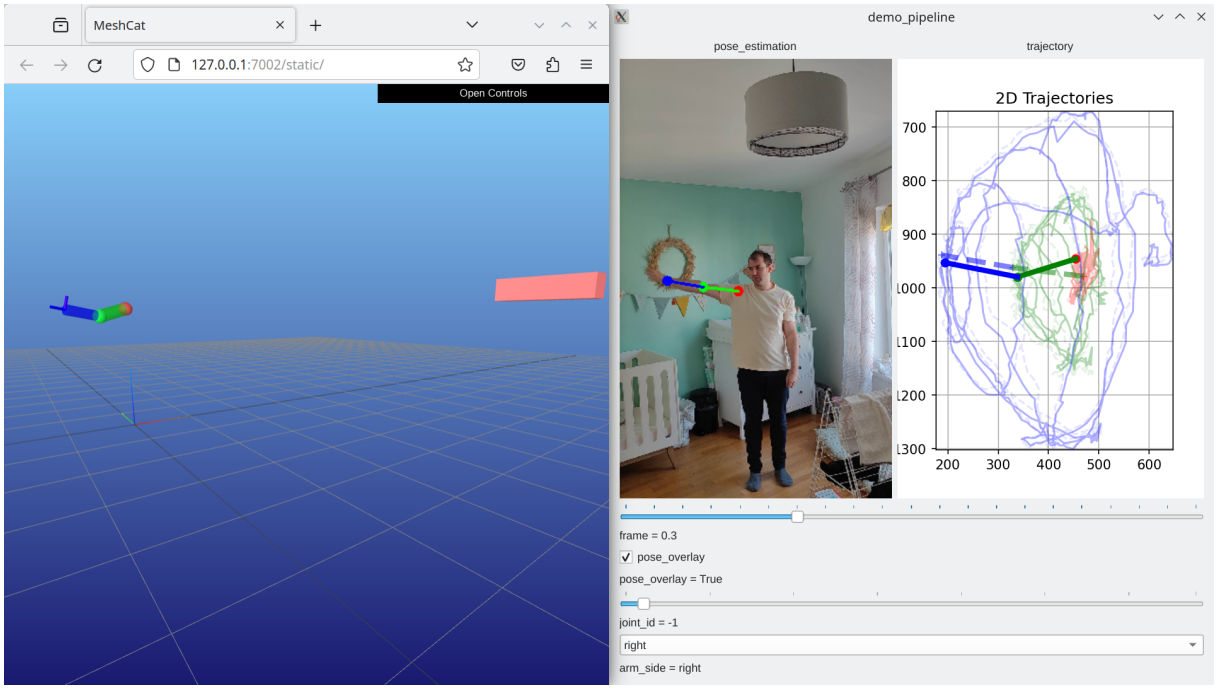

Figure 7: Demo of the pose estimation pipeline. Left: Digital twin of the arm is synchronized witg the GUI. Camera is indicated in pink. Right: graphical user interface to select a specific frame and visualize the 2D joints position (dashed - right curve) aswell as the 3D reprojection of the arm (plain - right curve).

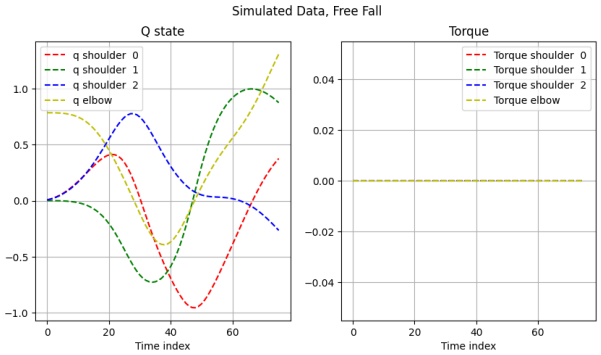

Figure 8: Simulated ground truth data of free fall.

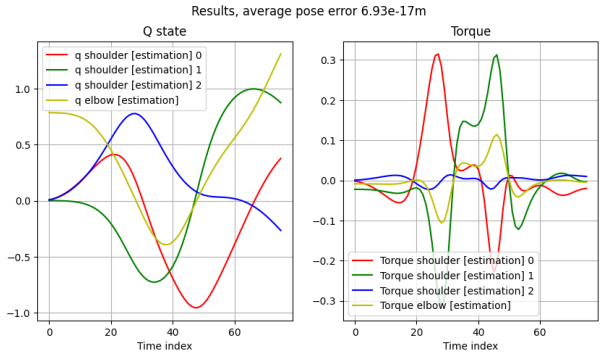

Figure 9: Torque estimation without regularization, showing erroneous non-zero torques despite accurate pose estimation.

3.7 Inverse Dynamics Results

Our initial tests of the optimizer were conducted on simulated data, enabling us to assess its performance and fine-tune the loss weights.

Free Fall, Simulated. We began with a simple free-fall scenario. The arm's initial configuration was horizontal, parallel to the ground, with an elbow angle of $\pi/4$ radians. This simulation was performed using the Runge–Kutta 2 method, and the resulting ground truth is depicted in fig. 8.

We fed the optimizer with the ground truth 3D poses and initial state vectors $q_1, \ldots, q_T$ and zero torques. Without torque regularization, the optimizer yielded physically implausible torque estimations, despite low pose error, as shown in fig. 9.

These inaccuracies are attributed to the central difference approximation for velocities and accelerations, and potential undersampling, which introduces errors that the optimizer compensates for with non-zero torques.

Incorporating torque regularization significantly improved the results. While the pose error marginally increased, it remained within a reasonable range of a few millimeters. Crucially, the torque estimates were much closer to the expected zero value, as illustrated in fig. 10.

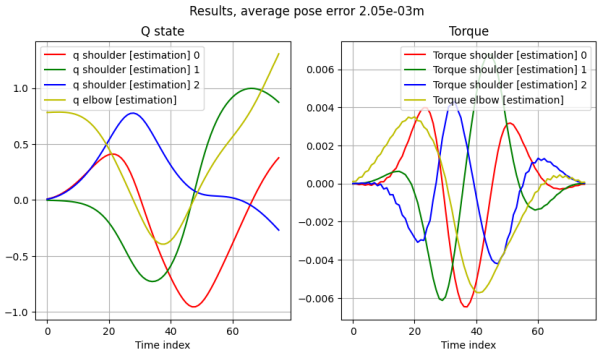

Figure 10: Torque estimation with regularization, showing a pose error still fairly low and torques that align much more closely with the ground truth.

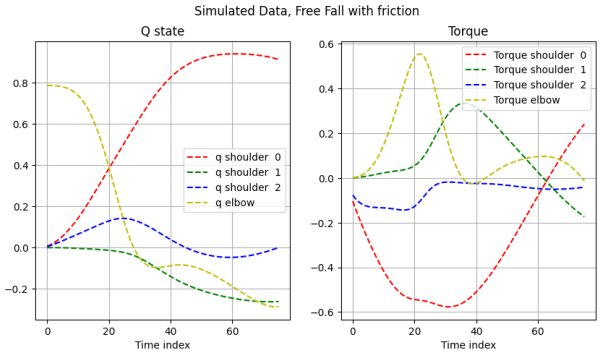

Figure 11: Simulated ground truth data of free fall with friction.

These findings highlight the importance of torque regularization in maintaining physical plausibility, especially when dealing with approximations issues.

Free Fall with Friction, Simulated. We tested the free fall with friction, represented by the equation $\tau_m = -0.1\dot{q}$. This test aimed to evaluate the optimizer's ability to reconstruct non-trivial torque values. Ground truth and results, shown in fig. 11 and fig. 12, indicate that while the results are not flawless, they demonstrate the optimizer's capability to reconstruct coherent torques.

Static Arm. An intriguing test scenario is the static arm, where the arm is maintained stationary in a plane parallel to the ground. The purpose of this setup is to recover the constant static torque that maintains the arm's position, given in fig. 13.

This static test uncovers another limitation within the optimization framework. It introduces a trade-off between the 3D consistency loss l_{3D} and the torque regularization l_{torque}. But if the torque is important, like in this case, this can result in significant errors in both the estimated position and the computed torque, as shown in fig. 14.

Real Data. When applying our optimizer to real-world data, we encountered several challenges. The optimization process in SciPy proved significantly slower on real data, with 2 seconds of video requiring up to 10 minutes of computation. Moreover, time constraints and the intricate nature of fine-tuning the loss weights for the optimizer prevented extensive testing on real data. Consequently, we could not obtain conclusive results from real-world experiments. We also understand why the authors of the original paper chose to compile the inverse dynamics optimizer program based on CERES and Pinocchio.

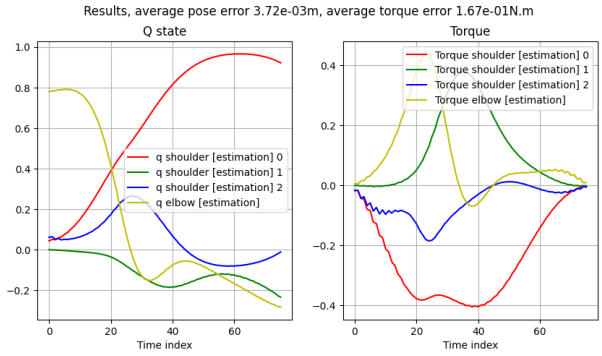

Figure 12: Results of the optimization on the free fall with friction.

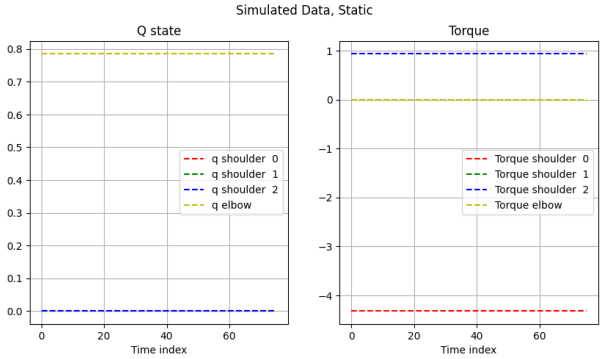

Figure 13: Simulated ground truth data of static arm.

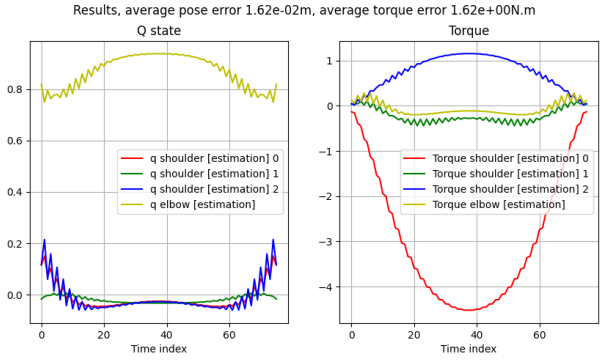

Figure 14: Results of the optimization on the static arm.

4 DISCUSSION ON THE PAPER

Our review and partial reimplementation of the work by Li et al. [7] have led us to several observations about the original study.

- **Method and Code Complexity:** The complexity of the approach, both in terms of the underlying methodology and the associated codebase, poses significant barriers to replication and extension.
- **Simplification of the Vision Pipeline:** The original study uses individual neural network models for each joint in contact recognition potentially leading to redundant feature extraction. A more streamlined and modern approach could employ a unified neural network (backbone) to handle these tasks more efficiently. The same remark can be made on the object endpoint detection.
- **Challenging Loss Coefficient Tuning:** Fine-tuning the loss coefficients in the optimization problem presents a significant challenge. The balance between data fidelity and regularization, requires careful calibration, which may limit the method's generalizability to diverse datasets. As there are no clear guidelines on how to tune these coefficients, one can assume that the authors fine tuned them to get correct accuracy on the Parkour dataset used for the quantitative evaluation.
- **Confidence in Force and Torque Estimations:** Assessing the accuracy of reconstructed forces and torques is problematic due to the limited benchmarks available for validation. The authors deserve credits for providing quantitative results (based on the Parkour dataset). Getting ground truth data for these quantities is very challenging. Realistic simulations do not seem possible.
- **Consistency in Object Weight and Body Dimension Priors:** In their method, weights and weight matrices are predefined. But the movement heavily depends on the weight and dimensions of the manipulated object and the human body, which can vary significantly across different videos.
- **Relevance of Full Body Dynamics:** Given the necessary approximations in velocity and acceleration, and the inability to strictly enforce full-body dynamics, the added value of attempting to reconstruct these dynamics as opposed to focusing solely on kinematics can be questionned. A comparative analysis with kinematics-focused methods could be insightful.
- **Lack of Data Adimensionality in Optimization:** The absence of adimensionalization or standardization in data processing could affect the model's ability to generalize. This issue becomes particularly evident in scenarios where similar movements are performed at different speeds, potentially leading to varied results. One would naturally divide the 2D reprojection errors by the image diagonal size (in pixels), the 3D reprojection errors by the length of the arm, the dynamics constraint (torque error) by the static torque of the arm at rest etc... The constants used to weight the cost terms would naturally become more consistent and easier to tune.
- **Torque Regularization:** As discussed in section 3.7, the use of $l_{\text{torque}} = \|\tau_m\|^2$ as a regularization term can lead to suboptimal behavior, particularly in situations where the motion requires significant torques. The simplest correction would be to penalize deviations from the torque obtained in a static configuration deduced from the current state $l_{\text{torque}} = \|\tau_m - \tau_m^{\text{STATIC}}\|^2$. Another more appropriate regularization might be on the derivative of the torque, $\dot{\tau}_m$, which would allow the model to accommodate necessary high torques while still smoothing the energy used in the movement.

5 CONCLUSION

In conclusion, our review and reimplementation of Li et al. [7]'s paper provided valuable insights into the complexity and challenges of estimating 3D motion and forces in human-object interactions from RGB videos. The original work stands as a significant novel contribution to this field. While our reimplementation is restricted to a simpler use case, we still faced a few difficulties, particularly in applying the method to real-world data. It still allowed us to highlight some limitations of the original method and to propose some improvements.

REFERENCES

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. 2023. *Ceres Solver.* https://github.com/ceres-solver/ceres-solver

[2] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. 2017. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 7291–7299.

[3] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. 2019. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII).* IEEE, 614–619.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision.* 2961–2969.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[6] Angjoo Kanazawa, Michael J Black, David W Jacobs, and Jitendra Malik. 2018. End-to-end recovery of human shape and pose. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 7122–7131.

[7] Zongmian Li, Jiri Sedlar, Justin Carpentier, Ivan Laptev, Nicolas Mansard, and Josef Sivic. 2019. Estimating 3d motion and forces of person-object interactions from monocular video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 8640–8649.

[8] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. 2019. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172* (2019).

[9] Zhengyou Zhang. 2000. A Flexible New Technique for Camera Calibration. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 11 (2000), 1330–1334. http://dblp.uni-trier.de/db/journals/pami/pami22.html#Zhang00

A APPENDIX

A.1 Camera calibration

Since we're working in a controlled environment (not in the wild, unlike the original paper), we calibrate a single camera once and for all,

- using a 7x10 printed checkerboard shot in various orientations
- using the OpenCV implementation of the Zhang's method [9].

Below, we detail the focal length estimation from camera specifications and make sure it matches with the calibration estimation. Xiaomi Mi11 Ultra main camera ($2.8\mu m$ pixel pitch) specifications in photo mode:

- 24mm focal length - full frame (24x36mm) equivalent
- Sensor size 4000x3000 = 12Mpix

We end up with a focal length for the photo mode of $f_{\text{pix}}^{\text{photo}} = 24mm * 4000px/36mm = 2666px$.

But since we're using a FullHD video mode with a crop factor of around 15% on each side, it is needed to rescale the focal length acordingly $f_{\text{pix}}^{\text{video}} = 2666px * 1.3 * \frac{1920px}{4000px} \approx 1664px$. Calibration method provides a estimated focal length of $1690px$ which is close enough to the specifications. We assume the camera to be a pinhole and neglect radial distortion.

A.2 Code description

The code is available at:

github.com/balthazarneveu/monocular_pose_and_forces_estimation

The code is written in Python and relies on a several external libraries:

- Pinocchio for kinematics and dynamics computations aswell as the arm model.
- Meshcat for 3D visualization.
- OpenCV for the camera calibration and the image processing.
- MoviePy wraps video processing.
- Google Mediapipe for 2D and 3D pose estimation.
- Scipy for the Levenberg-Marquardt optimization.
- batch-processing to process multiple video files in a systematic way.
- interactive-pipe to display a GUI with graphs and images and interact with sliders and keyboard. This library works with Matplotlib as the default graphical backend but PyQT/PySide is highly recommended for the demo.

To process a new set of videos, located in the `data` folder, run the following command:

```
python scripts/batch_video_processing.py
-i "data/*.mp4"
-o "out"
-A demo
```

When the GUI pops up, press F1 to get the help menu to learn about the shortcuts. Press F11 to display in full screen. Do not forget to click the hyperlink in the terminal to open the MeshCat viewer in your browser.

If you're using a different camera, you'll need to calibrate your camera intrinsics first. Capture a calibration video sequence using a 10x7 checkerboard (print it and stick it on a cardboard or display it on your screen).

```
python scripts/batch_video_processing.py
-i "data/camera_calibration_<cam_id>.mp4"
-o "calibration"
-A camera_calibration
```

Then you'll be able to process your pose videos using the new camera intrinsics, by simply specifiying the calibration file path:

```
-calib "calibration/camera_calibration_<cam_id>.yaml"
```

The core of the code for inverse kinematics and inverse dynamics is located in: src/projectyl/dynamics