École Polytechnique Fédérale de Lausanne

CHILI Laboratory

Semester project report :

Level up the interaction space with Cellulo robots

Students:
Lucas Burget

Teacher:
Pierre Dillenbourg

Supervisor:
Wafa Johal

June 7, 2019

Contents

Introduction

Cellulo, as tabletop tangible robots, are a great tool for interactive learning and upper limb rehabilitation. For the moment, the main way of interaction is to take them in hand and move them in the plane. Thus, due to the laws of physics, the different gestures induced are mainly constrained to be in 2D and the workspace is bounded by the length of the arm. More precisely, for upper limb rehabilitation, it limits the solicitation of certain arm joints, for instance the shoulder absduction/adduction. In order to overcome these limitations, the idea was to use two wristbands with sensors to consider the arms of the user as a remote control. For the rehabilitation part, this would allow the patient to train the joints of his arm with greater amplitude. But more generally, this new mean of interaction with the Cellulo allows more possibility in the creation of activities.

This semester project studied two different aspects: pointing direction and gesture recognition. The first could be used by the game Master to change the behavior of some robots in order to adapt to the situation. And the second could be destined to control a swarm of robots. The governing principle of this semester project was to implement systems as light and easy to use as possible.

Chapter 1

Pointing direction

1.1 Theoretical background

In order to know where the user is pointing at, its relative position to the workspace must be known. B. Gromov and al. [1, 2] presented a method to find the transformation between the robot's and the user's frame.

First, we define three reference frames (Figure 1.1.1):

- The operator's frame H, located at the operator's feet (x-axis: front, y-axis: left, z-axis: up)
- The workspace's frame W, located at one of the corners of the paper sheet (z-axis: down, other axes unknown)
- The sensor's frame S, located on the operator's wrist (z-axis: up, other axis unknown)

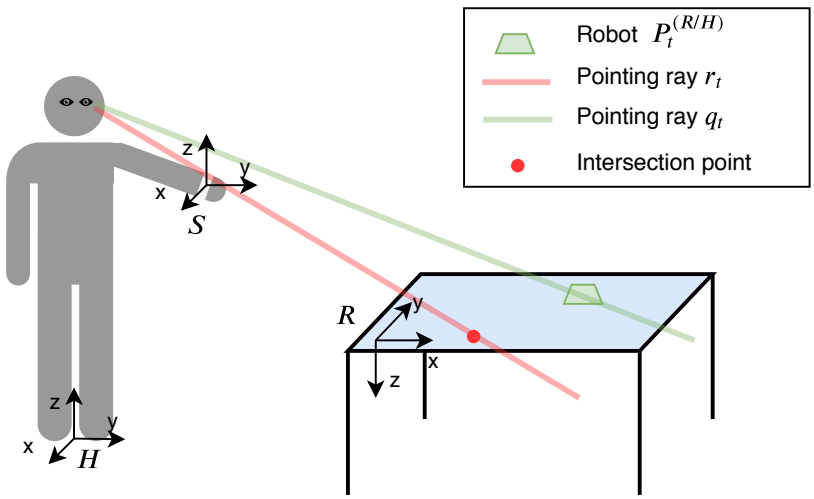


Figure 1.1.1: Schema of the setup

Then, we define a *pointing ray* r which represent the direction at which the user points. In our implementation, we used the *head-finger* model [3] which mean that this ray starts at the dominant eye and pass at the tip of the finger. To reconstruct it in H, we use the orientation from the IMU and measures of the user's body.

The calibration phase aims at finding the transformation T_R^H from H to R. In order to do so, the user points at the robot which moves to 4 known locations. It gives a set of pairs C of pointing ray $r_t^{(H)}$ and robot localization $P_t^{(R)}$, at the same time stamp t.
The transformation T^* converts the robot position in its own frame $P_t^{(R)}$ to the user's frame $P_t^{(H)}$. Using it, we can construct $q_t^{(H)}$, a ray that shares the same origin as $p_t^{(H)}$ but passes through the point $P_t^{(H)}$. Then we define the unsigned error angle $\theta_e(C, T^*)$ between $q_t^{(H)}$ and $p_t^{(H)}$ and we search the T^* that minimizes it over all pairs in C. The optimisation method chosen was *Nelder-Mead* because it is robust and the convergence speed is not a constraint.

1.2 Implementation

The MetaWear R+ (Mbientlab) sensor attached to the wrist was used to retrieve the arm orientation. It is composed of a 3-DoF accelerometer, a 3-DoF gyroscope, and a 3-DoF magnetometer and communicates via Bluetooth. Furthermore, it runs a sensor fusion algorithm which provides a real-time orientation estimation.

The ROS framework was used to communicate between the different parts: robots, IMU and software. There are mainly 3 calibration steps before we are able to guide the robot:

- **user's morphology estimation**: In order to implement the *head-finger* model, five different characteristics are needed: *eyes' height, shoulder-dominant eye vector, and should-finger length.* In order to automated this process, we used different models (*openpose*[4] and *PifPaf*[5]) which extract the skeleton of the user based on a picture (see Figure 1.2.1).
- **IMU calibration**: The user is asked to stretch his arm horizontally in front of him. The program then computes the *yaw* offset between S and H to align these frames.
- T_R^H **computation**:
  - The user is asked to point to the robot. This latter will move to 4 different points and the pairs C ($r_t^{(H)}$, $P_t^{(R)}$) are saved when it reaches its destination.
  - Based on the pairs C and the user's morphological characteristics, it finds $T^* = argmin_T \theta(T, C)$.

Finally, based on the T^* found earlier, the pointing ray r_t^H is translated in the robot's frame and the intersection I_t^R with its plane is computed.

1.3 Results and Discussion

In order to visualize the result, since it was not possible to represent the computed I_t^R on the real workspace, we used Rviz, a software that shows position with respect to different frames (Figure 1.3.1).

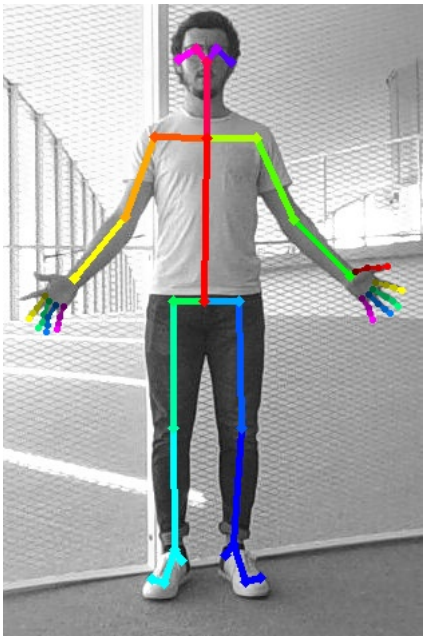


Figure 1.2.1: Skeleton extraction

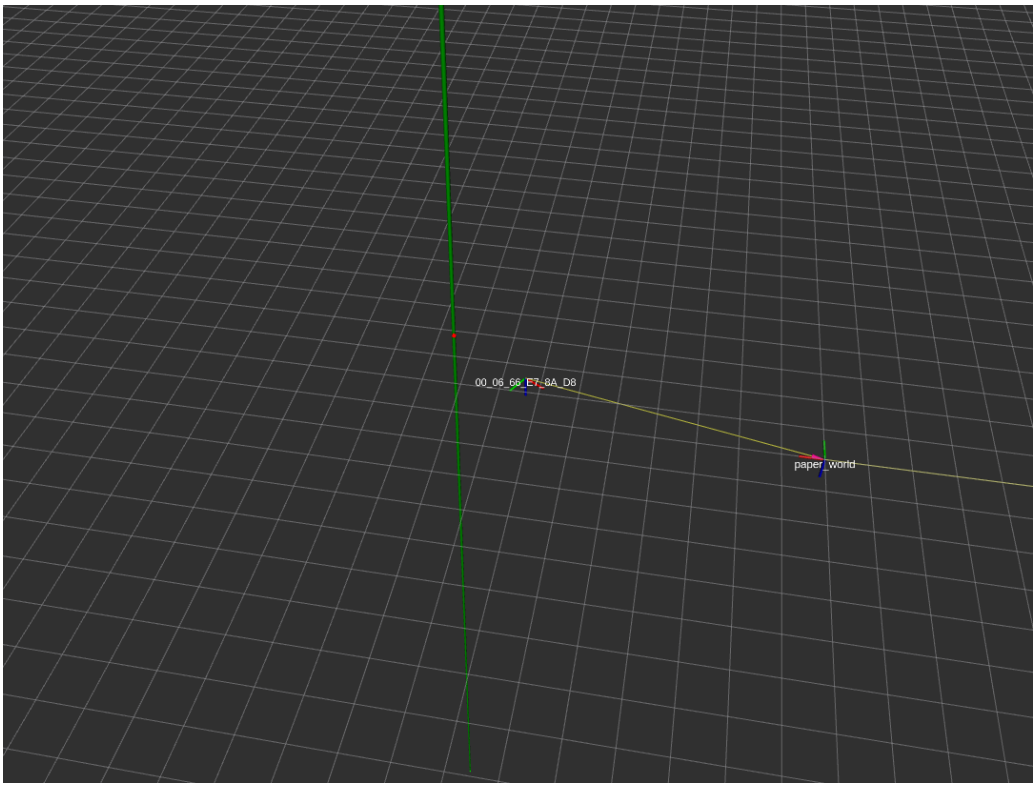


Figure 1.3.1: Rviz screenshot where we see the offset when the user points at the robot after calibration. *Paper_frame* is the origin of the robot reference frame. *00_06_66_E7_8A_D8* is the position of robots and the green line is the pointing ray.

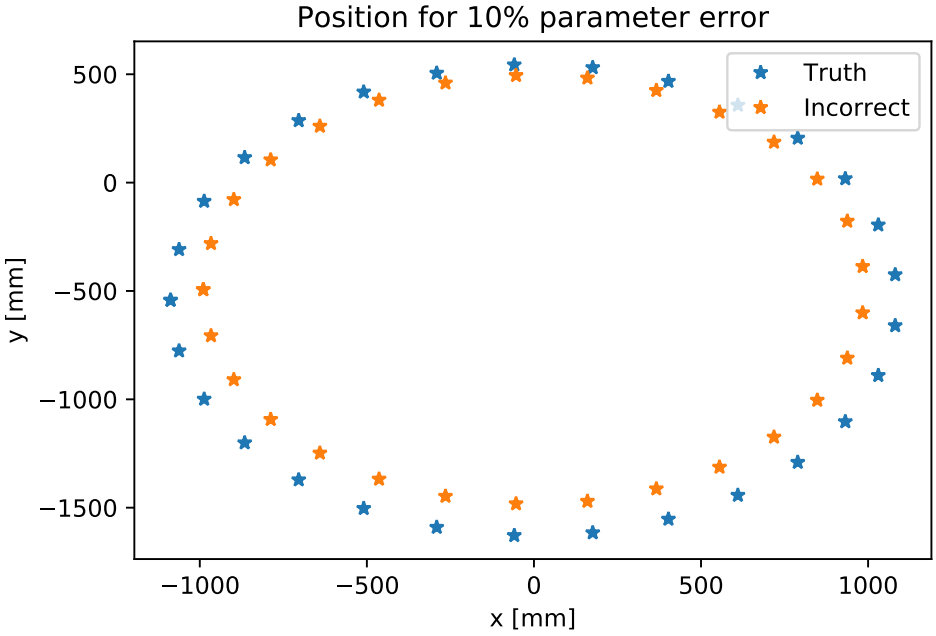
As we can see, even if we point a the robot, there is an offset. There are mainly three sources of error: bad estimation of user's morphology, IMU frame misalignment and obviously the intrinsic human limitation.
To see the impact of the two first, we simulate the data. All the functions used are the same as in the

real program. We consider that the localisation of the robot is accurate enough and don't need to be corrupted. The pointing rays come from an ideal user pointing at 45°to the ground, in a circle around himself. The ground truth points were then generated by taking the intersection of these rays with the ground plane. Then, the optimisation process was fed with incorrect pointing rays r_t and the correct P_t and so give an incorrect $\tilde{T}^*$. Finally, rays with the same error are transformed into the robot frame and the intersections are computed.
Two scenarios have been created:

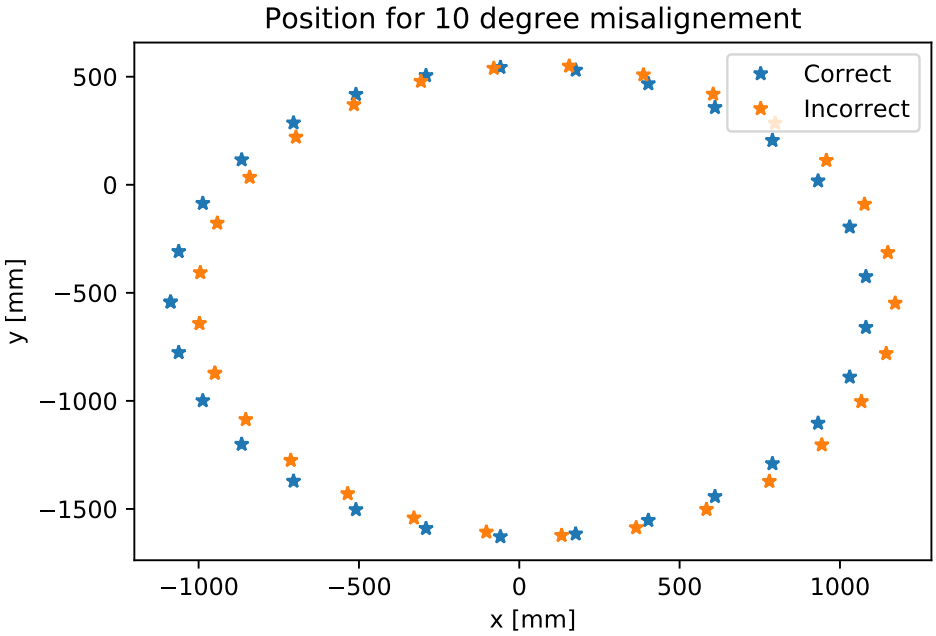
- **Angle error**: the $\tilde{r_t}$ fed to the optimisation process are shifted by an angle between 0 and 10 °with respect to r_t. In Figure 1.3.2b we can see the true position and the wrong estimated ones. As we can see in Figure 1.3.3b, even a small deviation can lead to tens of centimeters errors.
- **Morphological error**: the $\tilde{r_t}$ fed to the optimisation process are biased by a percentage of change in the morphological parameters. In Figure 1.3.2a we can see the true position and the wrong estimated ones and in Figure 1.3.3a we see the error depending on the variation in user's estimated parameters.

Furthermore, based on the user's feedback, we noticed a few things:

- None of the two packages used to estimate the human model were able to produce meaningful data. The best results were obtained by measuring the subject characteristics by hand.
- We need to take into account the height of the robot. The user tends to point to the center of the leds.
- Better results were obtained when pairs C were registered when the robot was static. This can be explained by the fact that the user has a small delay when following the robot, and so inducing an offset in the pairs C.
- As proposed by, Goromov et al [1], a feedback of the actual pointing ray could improve the precision. Indeed, it is complicated to visualise where we are pointing at. A small laser ray mounted on the wrist would be a trail to explore. In this case, the *forearm* model would be more suitable, since it defines a ray with the origin at the elbow and passing via the wrist joint.

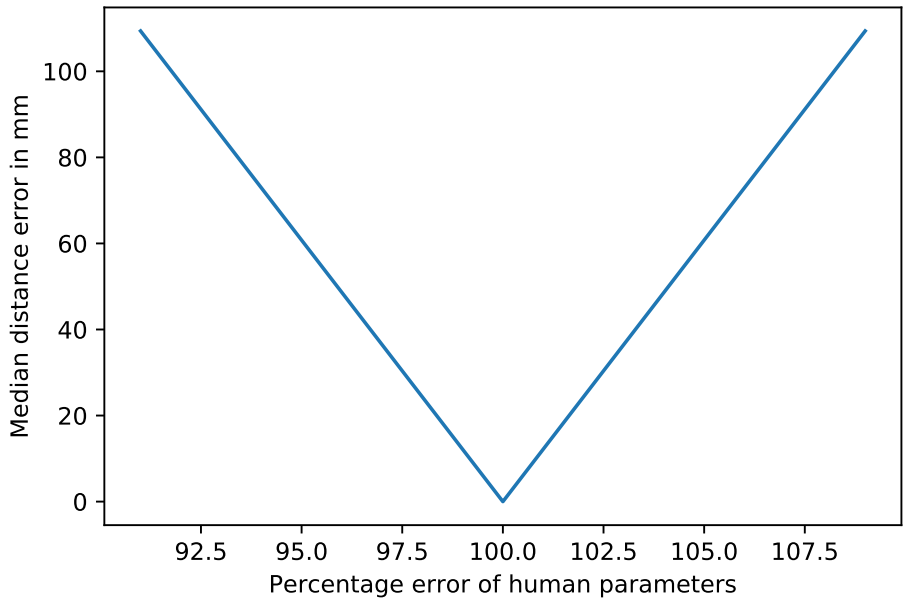


(a) Ground truth and estimated position for a 10% error on the human parameters.

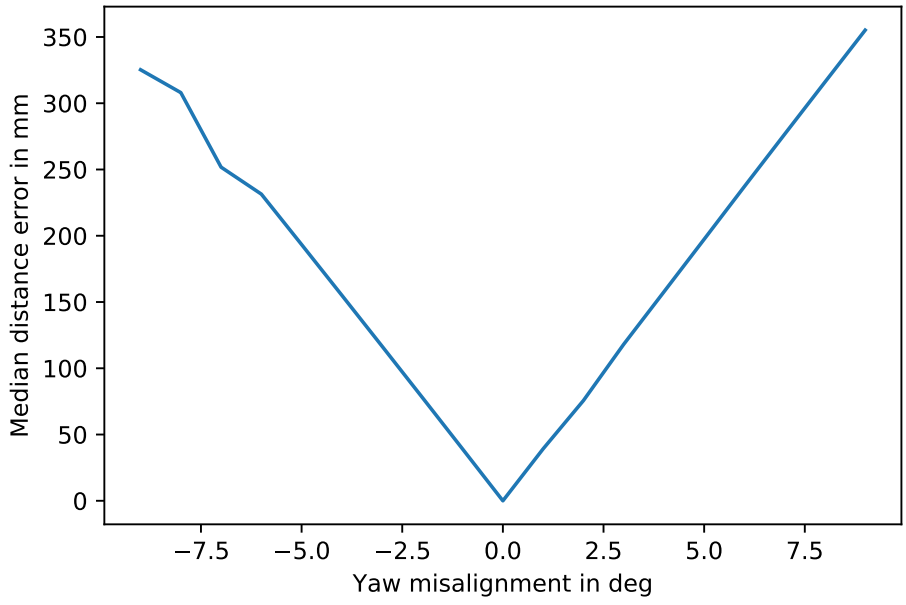


(b) Ground truth and estimated position for a 10°misalignment between H and S.

Figure 1.3.2: Position of ray $r_t^{(R)}$ intersection with the robot plane.



(a) Median distance error for different ratio of wrong human parameters.



(b) Median distance error for different angular misalignment between H and S.

Figure 1.3.3: Median distance error for two source of imprecision.

Chapter 2

Gesture recognition

2.1 Theoretical background

In this part, the goal was to train a model able to recognize gestures. This would permit to create fun games where a patient controls a swarm of robots with his arms. To allow to work efficiently and playfully, a set of varied gestures combining naturalness and amplitude has been selected. In Figure 2.1.1, there are a few of the gestures selected.

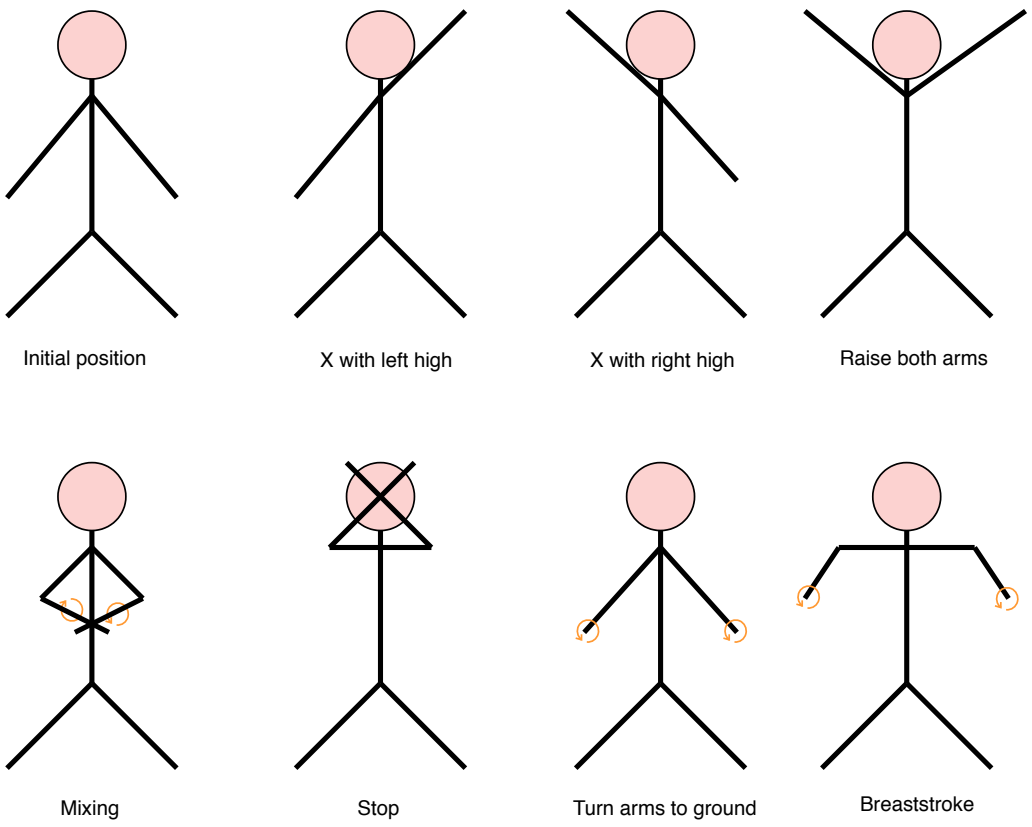


Figure 2.1.1: Initial position and 7 gestures

The model we will use is a 1D CNN network. It theoretically works well for identifying simple patterns within data which will then be used to form more complex patterns within higher layers. A 1D CNN is very effective when you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance.

This applies well to the analysis of time sequences of sensor data.

2.2 Implementation

We started by using the interface *metawear_ros* provided by B. Goromov to stream the data of the IMU. Unfortunately, we had to re-implemented it in order to connect multiple IMUs and to have a faster and more stable data transfer. The new interface publishes on respective ROS topics the acceleration, angular velocity, and orientation in quaternion obtained from the sensor fusion model of each wearable.

To collect the data for each gesture, another program was implemented. In order to ensure the reproducibility between the different users, is first asked to stretch horizontally both arms. Everyone will have the same reference frame with: *x-axis: points in front of the user, y-axis:points left, z-axis:points up.*
After this calibration, it starts to synchronize, with respect to time, the data from the 6 different ROS topics, 3 for each IMUs. Then it computes the linear acceleration (without the gravity), the angular velocity and the orientation around the new reference frame. In order to ease the record of the data, the user starts and stop the saving by double clicking. Each gesture is then stored into csv files that can be used later to train the model.

A few prepossessing steps are necessary to ease the classification. First, the data need to be sorted along time and down-sampled. It will reduce the noise and give equally spaced data. Then, the actual start of the movement is detected when $\frac{1}{6}\sqrt{\sum_{\forall acc} a_i^2}$ is higher than a threshold and from that moment we take a certain time window. Then, the acceleration and angular velocity are normalized between $[-1, 1]$ (no need for orientation since quaternions are already normalized). Finally, we compute the difference of value between two time step for all the data. This simplifies the work of the model since it has only to recognize the relative pattern and not the absolute one.

2.3 Results and discussion.

Unfortunately, due to a lack of time, I wasn't able to implement the model. I focused on creating clean tools that could be used for future work. The program collecting the data, once modified a bit, could be used later to feed the trained model in real-time and then implement the behavior of the robots.

The goal of this first step of recognition was to have a robust dictionary gestures. It would later be used to match with actions of the swarm. We defined a few combinations such as, *mixing* $\longrightarrow$ *random walks*, *stop* $\longrightarrow$ *stop moving* or *breaststroke* $\longrightarrow$ *Coverage of the area.* These associations are meant to be as intuitive as possible. In order to asses that these gestures are meaningful with respects to the desired behavior, experience with non-technical peoples needs to be performed.

Conclusion

The interactions with Cellulo based on gestures are complementary with the existing ones. For the first part of this semester project, with the setup used (only IMU on the wrist), the precision achieved is not sufficient to select robots nor to point to the exact location. The first feature could be achieved by adding laser for the user and receptor on top of Cellulo which would detect if the ray touch it, and then action could be performed. For the second feature, one could imagine pointing at predefined areas.
For the second, the tools to start to implement real-time gesture recognition are available and could be used to continue this project.

Acknowledgment

First, I want to thank Wafa Johal for her valuable advice and constant availability. I also thank Hala Khodr and Arzu Güneysu for their help throughout the project. Finally I would like to thanks Pierre Dillenbourg for giving me the possibility to do my semester project in a dynamic and friendly laboratory.

Bibliography

[1] B. Gromov, L. Gambardella, and A. Giusti, "Robot identification and localization with pointing gestures," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3921–3928, Oct. 2018.

[2] B. Gromov, G. Abbate, L. Gambardella, and A. Giusti, "Proximity human-robot interaction using pointing gestures and a wrist-mounted IMU," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, May 2019. to appear.

[3] S. Mayer, V. Schwind, R. Schweigert, and N. Henze, "The effect of offset correction and cursor on mid-air pointing in real and virtual environments," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, (New York, NY, USA), pp. 653:1–653:13, ACM, 2018.

[4] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *arXiv preprint arXiv:1812.08008*, 2018.

[5] S. Kreiss, L. Bertoni, and A. Alahi, "Pifpaf: Composite fields for human pose estimation," *CVPR, arXiv preprint arXiv:1903.06593*, 2019.