# Pointing at Moving Robots: Detecting Events from Wrist IMU Data*

Gabriele Abbate, Boris Gromov, Luca M. Gambardella, and Alessandro Giusti[1]

***Abstract*— We propose a practical approach for detecting the event that a human wearing an IMU-equipped bracelet points at a moving robot; the approach uses a learned classifier to verify if the robot motion (as measured by its odometry) matches the wrist motion, and does not require that the relative pose of the operator and robot is known in advance. To train the model and validate the system, we collect datasets containing hundreds of real-world pointing events. Extensive experiments quantify the performance of the classifiers and relevant metrics of the resulting detectors; the approach is implemented in a real-world demonstrator that allows users to land quadrotors by pointing at them.**

## I. INTRODUCTION

We consider a system composed of one or more (ground or flying) mobile robots and one or more nearby humans instrumented with a wrist-mounted inertial sensor, e.g. a smartwatch. In this context, we solve the problem of detecting the event that one of the humans is pointing at a moving robot with their straight arm, and following it for a short time.

We detect this event by matching the robot motion (measured by the robot's odometry module in its own fixed frame) to the corresponding arm motion (measured by the IMU in a different frame). Our approach is practical and easy to deploy as it does not rely on additional sensors, and does not assume that the humans or robots are co-localized with respect to each other or with respect to any external frame.

Many systems involving human operators and robots operating in the same space require the ability for an operator to easily address one specific robot in its line of sight, e.g. before providing a command or when calling for the attention of that robot. For example: in rescue scenarios [1] with several human rescuers and a team of flying robots [2], any rescuer in the team might need to address a specific quadrotor to assign it a specific mission or query its battery status; on a factory floor with multiple cooperating Autonomous Ground Vehicles and humans, an operator could notice one vehicle malfunctioning and need to immediately stop it.

Pointing at robots is a desirable and efficient interaction mechanism, because the act of pointing at physical entities is trivial and intuitive for humans (e.g. children naturally point at objects they are interested in or want to interact with [3]).

In general, detecting whether a human is pointing at a robot implies to: a) perceive the human's posture and—if they are pointing at something—a *pointing ray* in 3D space where the pointed object lies; b) determine whether the robot is close enough to the pointing ray.

Our system perceives the human's posture using only a wearable IMU sensor on the wrist (such as a smartwatch), which returns the wrist's 3D orientation. This raises two challenges: a) it is not easy to distinguish whether the operator is pointing at something or performing other unrelated movements; b) even if we know that the operator is pointing, the relative pose of the operator with respect to the robot is not known; it is therefore impossible to determine whether the robot is close to the pointing ray (and in case of multiple robots, which robot is being pointed at).

Our solution relies on the assumption that the target robot is moving and that the system is aware of the robot's trajectory, e.g. through the robot's odometry module. Then, we build upon our previous results [4] which describe how to recover the relative pose of the robot with respect to the operator, *assuming that the operator is pointing at the robot during a given period*; this is achieved by finding the relative transformation between the human and robot frames that minimizes, for any timestamp within the period, the distance between the robot position and the corresponding pointing ray.

Using this capability as a building block, this paper addresses the related but different problem of detecting whether an operator is pointing at a moving robot, given IMU readings and the robot's odometry. After reviewing related work (Section II) and formalizing the model (Section III) we describe and motivate our **main contribution** (Section IV): a learning-based solution that operates on features extracted from the input streams; we also compare an alternative approach operating on raw sensor data with a recurrent neural model. We experimentally evaluate their performance on real-world pointing datasets (which we release as supplementary material), and compare them to alternative approaches

*This work is partially supported by: the Swiss National Science Foundation (SNSF) through the National Centre of Competence in Research (NCCR) Robotics; and the European Commission through the Horizon 2020 project 1-SWARM, grant ID 871743

[1]Gabriele, Boris, Luca, and Alessandro are with the Dalle Molle Institute for Artificial Intelligence (IDSIA USI-SUPSI), Lugano, Switzerland. Email: {gabriele.abbate,boris,luca.gambardella,alessandrog}@idsia.ch.

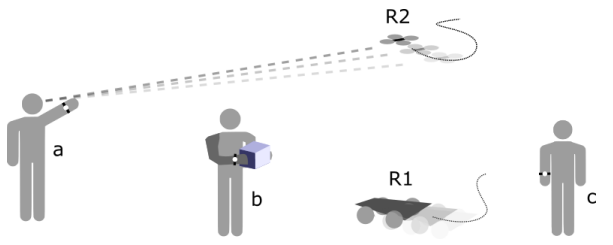


Fig. 1. Humans (a, b, c) equipped with IMU bracelets work in a shared space with robots (R1, R2); at any time, one operator (a) can select any moving robot (R2) by pointing at it for a few seconds. We propose a method to detect these pointing events.

and baselines; we extend and evaluate the approach on settings with multiple operators and multiple robots; finally, we deploy the system in a demonstrator where the user can immediately land a robot by pointing at it for a few seconds.

## II. Related work

Pointing gestures in robotics serve as an intuitive and natural control interface for waypoint navigation, pick-and-place tasks, for assessing joint human-robot attention, and for selecting individual and groups of robots. These tasks require the system to estimate a pointed direction (a pointing ray) and a pointed location using either external or wearable sensors.

A prevailing approach in robotics to estimate pointing rays relies on various vision sensors: monocular cameras [5, 6], stereo cameras [7], time-of-flight [8], and structured light depth cameras [9], placed either in a shared environment or directly on a robot.

The vision pipeline for pointing estimation [10] can be summarized in the following four steps: 1) localization of the human in 3D space relative to the robot; 2) segmentation of human body parts; 3) extraction of characteristic points (arm joints positions); 4) estimation of the pointed location.

On the contrary, using wearable inertial sensors requires: 1) reconstruction of the human body posture (arm joints positions); 2) estimation of the pointed location; 3) transformation of the estimated location to the robot's frame.

Pourmehr et al. [11] developed a system that estimates the pointing ray using a popular Kinect sensor (RGB-D camera) mounted on a ground robot. The sensor provides an easy access to the skeleton data, i.e. positions of body joints, in 3D space that is expressed in the camera coordinate frame. The authors compute the head-hand line (pointing ray) and check if it intersects with a virtual sphere placed at the sensor's origin to select the robot.

Selection of individual and multiple robots does not necessarily require localization of the user with respect to the robot. Nagi et al. [12] proposed a method based on binary-class ("pointing at me" and "pointing at somebody else") hand gesture classification. Each robot individually classifies the gesture and calculates the probabilistic scores for each class. The robots in the swarm then share their scores with each other to reach a swarm-level consensus on the robot being pointed at.

Once the system knows coordinate transformations between the user and the robots, pointing gestures can be efficiently used for agile control of fast-moving flying robots on complex trajectories [13] or for guidance of slow-moving ground robots [14], and for landing quadrotors on a precise spot [15].

## III. Model

### A. Definitions and pointing model

Consider a standing operator, that may (or may not) be pointing at a moving robot.

Let $\{H\}$ be a reference frame located at the operator feet, oriented in such a way that the $z$ axis points upward; let $\{R\}$ be an arbitrary fixed reference frame in which the robot's non-ideal odometry module measures its position, also oriented in such a way that the $z$ axis points upward. $\{H\}$ and $\{R\}$ are related by an unknown coordinate transformation $T^*$.

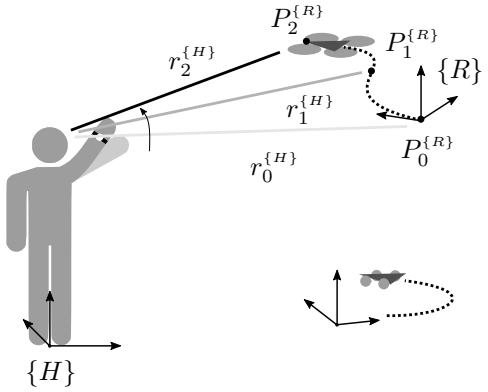


Fig. 2. A user pointing to a moving robot, which measures its motion in a fixed odometry frame $\{R\}$; another robot follows a different trajectory.

If the operator is pointing at something, the pointing ray $r$ is a 3D half-line on which the point that the human intends to indicate lies. We recover $r$ in frame $\{H\}$ using a simplified version of the eye-finger ray cast (EFRC) method by Mayer et al. [16], which defines $r$ as the half-line originating at the operator's eyes and passing through the tip of the pointing finger.

In particular, we assume that the operator is pointing with a straight arm, whose orientation in space is measured by a wrist-mounted IMU. We further assume that the eyes and the shoulder are vertically aligned, and that parameters of the operator body are approximately known (i.e. the shoulder and eyes height above ground, and the arm length). This allows us to compute the positions of the head and fingertip (and thus $r$) with respect to the operator coordinate frame $\{H\}$.

For a short period of time $\tau$, we collect the pointing rays $r_i^{\{H\}}$ in the reference frame $\{H\}$, and the corresponding robot positions $P_i^{\{R\}}$ in the frame $\{R\}$. This defines a set of pairs $\mathcal{C}$:

$$\mathcal{C} = \{(r_1^{\{H\}}, P_1^{\{R\}}), \ldots, (r_N^{\{H\}}, P_N^{\{R\}})\},$$

If the operator is pointing at the robot, we expect that the points $P_i^{\{R\}}$ lay close to their corresponding rays $r_i^{\{H\}}$.

In this case, a coordinate transformation $T^*$ between the two frames can be estimated using an optimization procedure [4], which we briefly summarize below.

### B. Relative Localization

For a given estimate $T$ of the unknown transformation $T^*$, we can convert the robot positions $P_i^{\{R\}}$ defined in the robot frame into the operator frame, i.e. $P_i^{\{H\}} = TP_i^{\{R\}}$. Using these points we define a new *robot ray* $q_i^{\{H\}}$ that shares the origin with the pointing ray $r_i^{\{H\}}$, but passes through the point $P_i^{\{H\}}$.

Now, we can define the error function $\theta$ for a set $\mathcal{C}$:

$$\theta\left(T, \mathcal{C}\right) = \frac{1}{N} \sum_{i=1}^{N} \angle(r_i^{\{H\}}, q_i^{\{H\}}) \tag{1}$$

where $\angle(\cdot) \in [0; \pi]$ represents the unsigned angle between the directions of two rays. The error function $\theta\left(T, \mathcal{C}\right)$ is therefore 0 if and only if all points lie on the respective ray, and positive otherwise.

We search for the coordinate frame transformation $T^*$ between the operator frame $\{H\}$ and the robot frame $\{R\}$ that minimizes the error function, i.e. that minimizes the average unsigned angle between all the pairs of vectors $r_i^{\{H\}}$ and $q_i^{\{H\}}$: $T^* = \arg\min_T \theta(T, \mathcal{C})$

The residual error $\theta(T^*, \mathcal{C})$ indicates how well the transformed robot positions fit the corresponding rays.

We express the transformation $T^*$ as a composition of a 3D translation $[t_x, t_y, t_z]$ and a rotation $\gamma_z$ around the $z$ axis; we can instead ignore rotations around $x$- (roll) and $y$-axis (pitch) because the $z$-axes of the operator and the robot coincide and correspond to the opposite direction of the gravity vector estimated by their IMUs. The problem is therefore reduced to that of finding a four-dimensional vector: $\rho = [t_x, t_y, t_z, \gamma_z]$ and can be handled with iterative nonlinear optimization solvers.

## IV. Learning-based pointing detection

A pointing detector is a function $D(\mathcal{C}) \rightarrow [0, 1]$ that: given as input a set of pairs, each composed by a pointing ray $r_i^{\{H\}}$ and the corresponding robot position $P_i^{\{R\}}$, collected for a given short period of time $\tau$; produces as output a scalar pointing score, i.e. the probability the operator was pointing at the robot $R$ during the whole period $\tau$. This output can be interpreted as the probabilistic prediction by a binary classifier with labels c0 (non-pointing) and c1 (pointing).

We expect a low score if the operator is not pointing at anything (e.g. standing still, walking, or actively working); crucially, we also expect a low score in case the operator is pointing at something that is not our robot $R$ (another robot, or something unrelated). Furthermore, we assume that the relative pose of $\{R\}$ w.r.t. $\{H\}$ is unknown.

This motivates the following observations.

- The target robot must be in motion; otherwise, it would be impossible to differentiate the case in which the operator is temporarily standing still (e.g. relaxing), from the case in which the operator is perfectly pointing at the non-moving robot. In fact, this configuration allows one to localize the robot on any point along the (static) pointing ray generated by the pose of the operator.
- If two robots follow trajectories with the same shape (i.e. there exists a similarity transform $S$ such that, for any time $t \in \tau$, the position of one robot can be obtained from the position of the other by applying $S$), it will not be possible to differentiate whether the user is pointing at one robot or at the other. This motivates our use of randomly-generated trajectories in experiments.

We describe and compare two detection approaches. One based on handcrafted features computed after relative localization is found; and one that operates on raw data in $\mathcal{C}$, independent on the relative localization approach above.

### A. Feature-based detector

First, the relative localization approach in Section III-B is applied to $\mathcal{C}$. This yields a set of robot positions $P_i^{\{H\}}$ (and corresponding robot rays $q_i^{\{H\}}$), expressed in the same frame as the pointing rays $r_i^{\{H\}}$.

For each timestep $i \in \tau$, we can now compute: a) the angle error, $e_{\text{ang},i} = \angle(r_i^{\{H\}}, q_i^{\{H\}})$; b) the position error $e_{\text{pos},i}$, i.e. the point-line distance between $P_i^{\{H\}}$ and $r_i^{\{H\}}$; c) the elevation and azimuth of ray $r_i^{\{H\}}$; d) the elevation and azimuth of ray $q_i^{\{H\}}$.

From these, we extract the following features and use them in a supervised classification approach.

- F1: mean and standard deviation of $e_{\text{ang},i}$ for $i \in \tau$;
- F2: mean and standard deviation of $e_{\text{pos},i}$ for $i \in \tau$;
- F3: the Pearson correlation coefficient between the rate of change of the elevation of $r_i^{\{H\}}$, and the rate of change of the elevation of $q_i^{\{H\}}$.
- F4: the Pearson correlation coefficient between the rate of change of the azimuth of $r_i^{\{H\}}$, and the rate of change of the azimuth of $q_i^{\{H\}}$.

In case the operator is indeed pointing at the robot, we expect the average error $e_{\text{ang},i}$ to be low (in fact, this means that the robot positions align well with the pointing rays). However, if the operator is still (but the robot moves), the relative localization approach will still perfectly minimize $e_{\text{ang},i}$ by localizing the robot infinitely far along the direction of the static pointing ray. This motivates the choice of F2, because $e_{\text{pos},i}$ would penalize this event.

F3 and F4 capture different aspects with respect to F1 and F2: in particular, we expect them to capture whether the motion of the pointing ray matches (in direction and speed, not necessarily in magnitude) with the motion of the robot ray. Because they capture the consistency of motion rather than a geometric alignment, we expect that these features represent complementary information with respect to F1 and F2—a hypothesis we verify in Section V.

Using a concatenation of F1, F2, F3 and F4 (6 total features) to represent an instance, we train a Random Forest classifier to predict whether the instance has class c0 or c1.

### B. Raw-data detector

As an alternative approach, we train a detector that does not rely on the relative localization procedure nor on handcrafted features. In particular, we consider a sequence-to-scalar recurrent neural network based on a single LSTM cell stacked with a fully connected layer (42k total parameters). The model takes as input two data streams: a stream of raw IMU data (which, in the feature-based approach, was used to reconstruct wrist joint movements); and a stream of robot odometry information. Those streams are synchronized at the lowest frequency available (20 Hz from the drone odometry) and combined to produce a 13 features vector (positions and

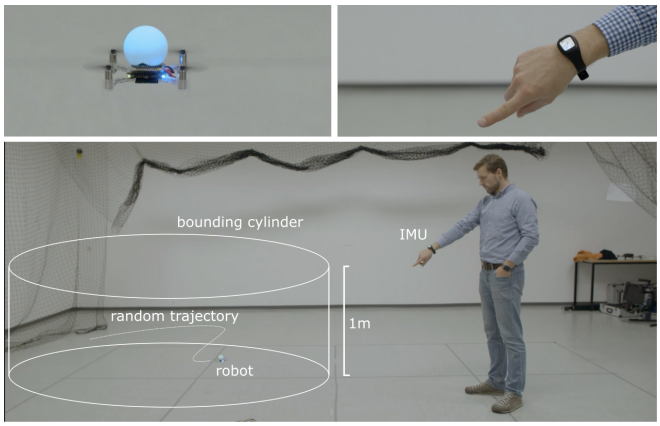


Fig. 3. Data acquisition setup.

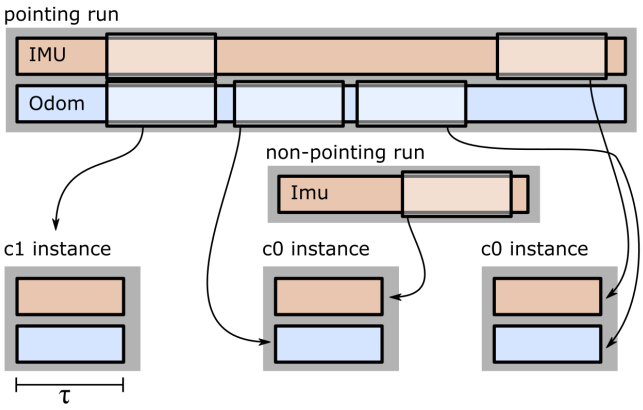


Fig. 4. Generating datasets from collected data.

velocities for the robot over 3 axes; 3D acceleration and orientation as quaternions for the IMU). At each timestep, the model outputs the pointing score. LSTM networks have many successful applications in Human Activity Recognition tasks based on wearable data [17, 18] and can capture complex temporal patterns in the input signals.

## V. Experimental Setup

### A. Hardware

Experiments are conducted in an indoor flying arena using a nano-quadcopter (Bitcraze Crazyflie 2.0 [19]), with onboard visual odometry using a Bitcraze Flow v2 deck equipped with a downward-looking optical flow sensor (PMW3901); the drone accepts waypoints via ROS interface and streams its position at a rate of 20 Hz in an arbitrary fixed odometry frame.

The operator wears an inexpensive IMU bracelet (Mbientlab MetaWearR+ [20]) that has the form factor of a wrist smartwatch. The device is equipped with a 3-DoF accelerometer, 3-DoF gyroscope, and 3-DoF magnetometer; readings are fused onboard to output an accurate estimation of the wrist's 3D-orientation in an arbitrary fixed reference frame whose z-axis points upward. The data is streamed to the host PC at a rate of 50 Hz rate via a Bluetooth 4.0 link.

### B. Data collection

We fly the drone on random trajectories generated as follows. First, the drone is manually flown to a position far from any obstacle; then, we sample a sequence of 3D points randomly distributed within a cylinder (radius of 1.5m, height 1m) centered at the drone's initial position. At random intervals (uniformly distributed between 1 and 3s), we send the next waypoint in the sequence to the drone that would override the previous target. This generates random 3D trajectories which stay confined in a limited area. The drone controls and measures its pose using the onboard visual odometry module.

We collect synchronized IMU and odometry data while the drone follows its trajectory. The user stands in place and points at the drone with a straight arm. This way we collected 20 runs ("pointing runs") each 1 minute long with different users, that were standing at random positions relative to the drone and at a distances ranging from 1 to 15 meters.

We also collect "non-pointing runs" totalling 40 minutes, containing IMU data sampled while users perform various tasks (walking, gesturing, moving boxes...), including occasionally pointing at various objects. These runs do not contain any trajectory information.

### C. Dataset Generation

We use half of the runs (both pointing and non-pointing) for training (30 minutes total), and the remaining for testing (30 minutes).

For a given window length $\tau \in \{1, 2, 3, 4, 5, 8\}$ seconds, we generate one training dataset (using all training runs) and one testing dataset (using all testing runs). Each dataset is composed by 5k c0 instances, and 5k c1 instances. Each instance is a sequence of $20 \cdot \tau$ pairs $(P_i^{\{R\}}, r_i^{\{H\}})$ sampled at 20 Hz from the runs, as follows.

As shown in Figure 4, c1 instances are built by sampling $\tau$ seconds of synchronized IMU and odometry data from a pointing run. Half of c0 instances (2.5k per dataset) are built by sampling odometry data from a pointing run, and IMU data from a non-pointing run. The other half of c0 instances are built by sampling odometry data from a pointing run, and IMU data from a *different* pointing run (or the same run at a different time). The former half represents instances in which a user is not pointing at a robot. The latter represents instances in which a user is in fact pointing, but at a different robot (which is following a trajectory not compatible with the user's motion).

### D. Implementation and model training

The feature-based detectors are implemented using the Scikit-learn [21] library. The nonlinear optimization problem for relative localization is solved with the quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno [22] as implemented in the `optimize.minimize` function from the SciPy library [23], with default parameters.

The LSTM detector is implemented and trained with the pytorch [24] library.

| Model | $\tau = 1$ | 2 | 3 | 4 | 5 | 8 |
|---|---|---|---|---|---|---|
| Angle residual [4] | 0.68 | 0.74 | 0.79 | 0.82 | 0.84 | 0.85 |
| Angle residual RF | 0.70 | 0.77 | 0.81 | 0.85 | 0.86 | 0.86 |
| F1+F2 | 0.80 | 0.86 | 0.89 | 0.92 | 0.94 | 0.94 |
| F1+F2+F3+F4 (full) | 0.89 | **0.93** | **0.94** | **0.96** | **0.96** | **0.97** |
| LSTM | **0.90** | **0.93** | **0.94** | 0.93 | 0.94 | 0.93 |

TABLE I

AUC VALUES ON THE TESTING SET FOR DIFFERENT MODELS. THE MODEL USED FOR FURTHER EXPERIMENTS IS MARKED IN BLUE.

The code for replicating our experiments is released as supplementary material.

## VI. EXPERIMENTAL RESULTS

We first quantify and compare the performance of the different classification approaches on the test set. Then, we apply the approaches on a sliding window to assess their performance when used for detection. Finally, we extend our results to the multi-operator multi-robot case, and describe the real-robot demonstrator shown in the supplementary video.

### A. Performance on the test set

We evaluate each classification approach by training the model on the training set, and running inference on all instances of the testing set. This generates a pointing score for each testing instance, for which we also have the ground truth class. We repeat the experiment for different lengths of the window $\tau$.

We quantify the overall performance of each binary classifier using the Area Under the ROC Curve (AUC) metric. One important advantage of the AUC metric is that it operates directly on the output score, and does not depend on the choice of a threshold; the AUC value can be intuitively interpreted as the probability that a randomly-chosen c1 instance has a score higher than the score of a randomly-chosen c0 instance. It follows that a AUC of 1.0 indicates a ideal classifier (since there must exist a threshold that separates c1 from c0 instances), whereas a AUC of 0.5 indicates a uninformative classifier. For a given threshold $t$, we also compute the accuracy (ACC); the False Negative Rate (FNR), i.e. the fraction of c1 instances that score lower than $t$; and the False Positive Rate (FPR), i.e. the fraction of c0 instances that score higher than $t$.

Table I compares the AUC values between different classification approaches, for different window lengths $\tau$. We observe that the detection approach proposed in [4], i.e. using a threshold on the angle residual, performs poorly in all cases; using a more powerful Random Forest classifier on the same feature does not significantly improve performance, which motivates the need for further features; we confirm that F1+F2 are indeed informative, and that F3+F4 carry useful complementary additional information. In the following experiments, we focus on the full model (F1+F2+F3+F4) with $\tau = 5$ s.

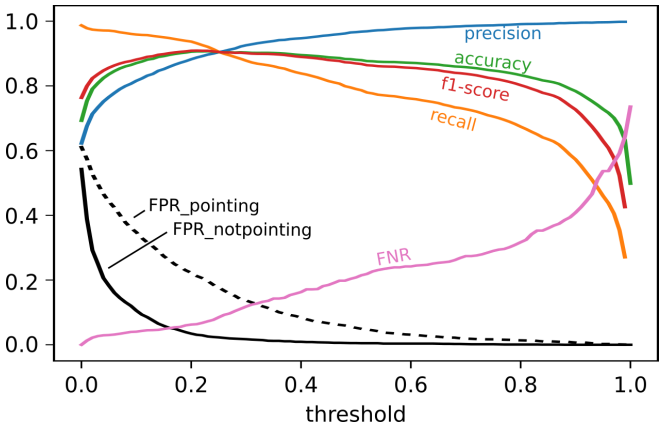


Fig. 5. Metrics as a function of the threshold for the full model with $\tau = 5$ seconds.

Interestingly, the LSTM model outperforms the handcrafted approach on short windows, but not on long windows. This could be caused by the fact that the relative localization approach (on which the handcrafted approach relies) yields more relevant and robust results for long windows than for short ones; on long windows, the explicit geometric modeling is therefore more valuable, and the LSTM is unable to capture the same information. In contrast, on short windows the LSTM might be advantaged by its ability to capture other cues for matching arm and robot motions (such as acceleration patterns) that are not represented in the handcrafted approaches.

Figure 5 considers the full model with $\tau = 5$ and reports various metrics as a function of the choice of the threshold. In particular, we separately report the FPR for c0 samples in which the user was pointing at a different robot; and the FPR for c0 samples in which the user was doing something different than pointing. We observe that the former type of c0 samples are harder (i.e. cause more false positives) than the latter.

### B. Application to pointing detection

In order to detect events in a stream of IMU and odometry data, we follow a *detection-by-classification* approach; for every new pair of odometry and IMU readings we receive (i.e. at a rate of 20 Hz), we apply the classifier to a sliding window containing the last $\tau$ seconds of data, which yields a pointing probability; if this value exceeds a threshold, we detect a pointing event. In case we have multiple operators and/or multiple robots, we apply the classifier to every robot-operator pair.

Figure 6 reports data from an experiment lasting 30 seconds with one operator and two robots, each flying a different random trajectory; during the first 15 seconds, the operator is not pointing at any robot. At $t = 15$, the operator starts pointing at one of the two robots. We apply the full model with $\tau = 5$ s to generate pointing probabilities for each of the two robots. We observe outputs generated at a given time $t$ correspond to a window at time $[t - 5, t]$. Shortly after $t = 15$, the pointing probability for both robots

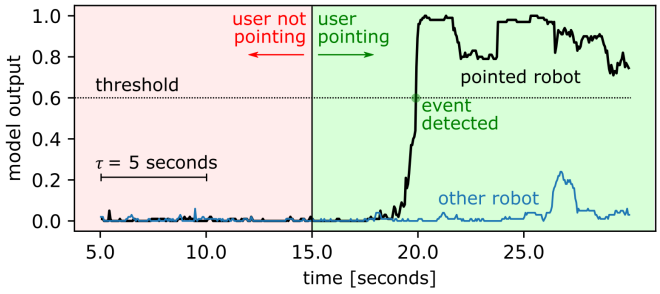


Fig. 6. Pointing probabilities for one operator and two robots; the operator starts pointing at one robot (black) at $t = 15$.

is still low, because only part of the sliding window contains actual pointing data. At time $t = 20$, the entire window contains valid pointing data, and the output for the pointed robot exceeds the threshold; this generates a pointing event. We observe that the output for the other robot stays well below the threshold.

### C. Pointing detection performance

We now study the performance of pointing detection for different values of the detection threshold. In particular, we are interested in quantifying: a) the delay before the pointing event is detected; and b) the probability that the event refers to the correct robot, in case two robots are considered.

To do so, we consider 1500 (partly-overlapping) 30 second intervals randomly extracted from the training runs. For each interval, we sample the IMU data, and *two* odometry streams; one from the robot that was being pointed; and one from a different run – representing a robot flying a different trajectory.

For each interval, we apply the trained model (full, $\tau = 5s$) on a sliding window; this yields a sequence of $(30 - 5) \cdot 20 = 500$ pointing probability values when considering the correct odometry; and a different sequence of 500 values when using the wrong odometry.

Then, for each interval and for each choice of threshold, we compute the "delay to detection" metric as the end time of the first window which exceeds the threshold when using the correct odometry stream. For example, if the first element of the sequence already exceeds the threshold, the total delay is 5.0 + 0.0 seconds. If the first 20 elements are below the threshold but the 21th exceeds it, the total delay is 5.0 + 1.0 seconds. Table II reports the mean delay to detection over all 1500 intervals for three threshold values We observe that raising the threshold, increases the mean delay to detection.

For each interval and threshold, we also compute whether the identified robot is the correct one. To do so, we check if the first pointing probability that exceeds the threshold is in the sequence using the correct odometry, or in the sequence using the wrong odometry. In case of a tie, we choose the option with the higher probability. Table II reports the robot identification accuracy, i.e. the percentage of the 1500 intervals in which the correct robot would be identified among the two.

| | Threshold | | |
|---|---|---|---|
| | 0.50 | 0.55 | 0.60 |
| Mean Delay to Detection [s] | 5.0 + 1.7 | 5.0 + 2.6 | 5.0 + 2.7 |
| Robot Identification Accuracy | 79.8% | 93.4% | 98.3% |

TABLE II
DETECTION METRICS FOR THREE VALUES OF THE THRESHOLD

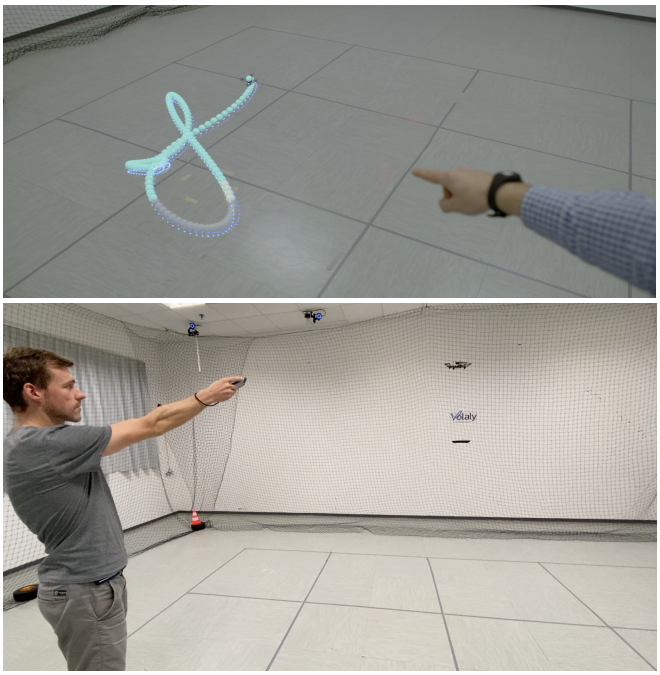

Fig. 7. Top: first person view while following a complex trajectory; bottom: application example with different hardware (handheld smartphone as IMU, DJI Tello quadrotor).

### D. Demonstrator

The supplementary video shows our demonstrator, in which a drone (Figure 7 top) follows a random trajectory in a confined area indefinitely; any user equipped with an IMU bracelet, from any nearby location, can cause the drone to immediately land by pointing at it for a few seconds.

## VII. Conclusions

We presented a practical approach for detecting the event that an operator points at a moving robot, based on learned classifiers operating on short windows containing paired streams of odometry and wrist orientation data. We collected datasets containing hundreds of real-world pointing events with matching IMU and odometry data, which we used to validate our approach. Extensive experiments quantify the performance of the classifiers and relevant metrics of the resulting detectors; we also show a real-world demonstrator. Videos, data and code is released as supplementary material.

## References

[1] J. Delmerico, S. Mintchev, A. Giusti, B. Gromov, K. Melo, T. Horvat, C. Cadena, M. Hutter, A. Ijspeert, D. Floreano, L. M. Gambardella, R. Siegwart, and

D. Scaramuzza, "The current state and future outlook of rescue robotics," *Journal of Field Robotics*, pp. 1–21, Aug. 2019.

[2] J. Cacace, A. Finzi, V. Lippiello, M. Furci, N. Mimmo, and L. Marconi, "A control architecture for multiple drones operated via multimodal interaction in search & rescue mission," *SSRR 2016 - International Symposium on Safety, Security and Rescue Robotics*, pp. 233–239, 2016.

[3] G. Butterworth, *Pointing: Where language, culture, and cognition meet*. Manwah, NJ: Lawrence Erlbaum Associates, 2003, ch. Pointing is the royal road to language for babies, pp. 9–33.

[4] 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)*, Oct. 2018, pp. 3921–3928.

[5] M. Pateraki, H. Baltzakis, and P. Trahanias, "Visual estimation of pointed targets for robot guidance via fusion of face pose and hand orientation," *Computer Vision and Image Understanding*, vol. 120, pp. 1–13, 2014.

[6] M. Monajjemi, S. Mohaimenianpour, and R. Vaughan, "UAV, come to me: End-to-end, multi-scale situated HRI with an uninstrumented human and a distant UAV," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, no. Figure 1. IEEE, oct 2016, pp. 4410–4417.

[7] K. Nickel and R. Stiefelhagen, "Pointing Gesture Recognition based on 3D-Tracking of Face , Hands and Head Orientation Categories and Subject Descriptors," *Proceedings of the 5th international conference on Multimodal interfaces*, pp. 140–146, 2003.

[8] D. Droeschel, J. Stückler, and S. Behnke, "Learning to interpret pointing gestures with a time-of-flight camera," *Proceedings of the 6th international conference on Human-robot interaction - HRI '11*, pp. 481–488, 2011.

[9] A. Cosgun, A. J. B. Trevor, and H. I. Christensen, "Did you Mean this Object?: Detecting Ambiguity in Pointing Gesture Targets," in *HRI'15 Towards a Framework for Joint Action Workshop*, 2015.

[10] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," *Ro-Man, 2012 Ieee*, pp. 411–417, 2012.

[11] S. Pourmehr, V. Monajjemi, J. Wawerla, R. Vaughan, and G. Mori, "A robust integrated system for selecting and commanding multiple mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2874–2879, 2013.

[12] J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro, "Human-swarm interaction using spatial gestures," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 3834–3841.

[13] 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, pp. 8084–8091.

[14] M. Tölgyessy, M. Dekan, F. Duchoň, J. Rodina, P. Hubinský, and L. Chovanec, "Foundations of Visual Linear Human–Robot Interaction via Pointing Gesture Navigation," *International Journal of Social Robotics*, vol. 9, no. 4, pp. 509–523, 2017.

[15] B. Gromov, L. Gambardella, and A. Giusti, "Guiding quadrotor landing with pointing gestures," in *Human-Friendly Robotics 2019*, F. Ferraguti, V. Villani, L. Sabattini, and M. Bonfè, Eds. Cham: Springer International Publishing, Feb. 2020, pp. 1–14.

[16] 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," *Proc. of the 2018 CHI*, 2018.

[17] Y. Guan and T. Plötz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–28, 2017.

[18] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[19] Bitcraze. The crazyflie nano quadcopter. https://bitcraze.io.

[20] Mbientlab. Wearable technology for healthcare. Mbientlab official web-page. https://mbientlab.com/.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] J. Nocedal and S. Wright, *Quasi-Newton Methods*. New York, NY: Springer New York, 2006, pp. 135–163.

[23] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-

Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.