Probabilistic Fusion of Persons' Body Features: the Mr. Potato Algorithm

Séverin Lemaignan
severin.lemaignan@pal-robotics.com
PAL Robotics
Barcelona, Spain

Lorenzo Ferrini
lorenzo.ferrini@pal-robotics.com
PAL Robotics
Barcelona, Spain

ABSTRACT

Multi-modal social perception usually involves several independent software modules, detecting for instance faces, voices, body skeletons. Those features need then to be matched to each other, to create a complete model of a person. While the problem is simple in one-to-one interactions, multi-party interactions require to optimize a probabilistic graph in order to find the most likely persons–features associations, while ensuring practical properties like stability over time. This paper presents an open-source algorithm that searches over all possible partitions of the relationship graph to identify the best partition. We playfully call this algorithm *Mr. Potato*, after the eponymous children' game.

CCS CONCEPTS

• **Computer systems organization** → *Robotics*.

KEYWORDS

Social representation, human-robot interaction

ACM Reference Format:
Séverin Lemaignan and Lorenzo Ferrini. 2024. Probabilistic Fusion of Persons' Body Features: the Mr. Potato Algorithm. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24), March 11–14, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3610977.3637479

1 INTRODUCTION

Robots deployed in social environments often perform multi-modal social perception. Multiple software module run in parallel and specialize in the detection of specific social features: faces, bodies (e.g., 3D skeletons), voices, etc.

Often, however, these features have to be combined and associated to form complete *persons*. The association might be direct (e.g., a facial recognition software module might directly associate a face to a specific person), or indirect (one might associate a body to a face based on the overlapping regions of interest in the source image, and transitively associate the body to a person).

The interplay between these *features* (in this paper, we primarily consider faces, bodies and voices) and their associated *persons*, has been recently formalised in the ROS4HRI standard [1], published as the ROS REP-155[1].

The ROS4HRI standard suggests to refer to features and persons through unique identifiers (that might be long-lived in the case of persons' identifiers – like the person's name –, or short-lived in the case of the face, body, voice identifiers, as, without tracking, the detectors for those features might have to assign new IDs every time the feature is (re-)detected).

The ROS4HRI standard also formalises a mechanism to broadcast possible associations between features and/or persons, with their corresponding likelihoods. For instance, a facial recognition node might broadcast a message like [{`john`, `face_432`, 0.8}, {`jane`, `face_432`, 0.2}] to indicate that a detected face has 80% chance of being John, and 20% chance of being Jane.

Over time, the possible associations published by the social perception modules of the robot span a probabilistic graph of relationships between social features and persons. The challenge is then to compute the most likely person–features associations, also accounting for the fact that the persons 'owning' the feature might yet be unknown (for instance, a module detects a face, but the person is not yet recognised).

This paper presents an algorithm (and its implementation) that compute such probabilistic associations. It is deployed on our social robots, and used on a daily basis to combine in real-time the various social features of humans in the robot' vicinity.

We playfully call this algorithm the 'Mr Potato' algorithm after the eponymous children' game where players have to put together a little potato-like character from all its body parts.

2 MOTIVATING EXAMPLE

Diagram 1 represents a possible *persons–features graph*, built over time be receiving probabilistic 'matches' between features pairs, or features and persons.

In this particular case, likelihoods of association between faces and persons are generated by a face identification node (like ROS4HRI's `hri_face_detect`[2]; associations between faces and bodies are computed by a face-body matching algorithm (like ROS4HRI's `hri_face_body_matcher`, that relies on overlapping regions of interest); associations between bodies and faces could be computed with a body recogniser (using, for instance, clothing colour matching); and voice–body association could be computed by yet another module, matching detected bodies with voices' direction of arrival.

HRI '24, March 11–14, 2024, Boulder, CO, USA

ACM ISBN 979-8-4007-0322-5/24/03…$15.00
https://doi.org/10.1145/3610977.3637479

[1] https://www.ros.org/reps/rep-0155.html
[2] https://github.com/ros4hri/hri_face_detect

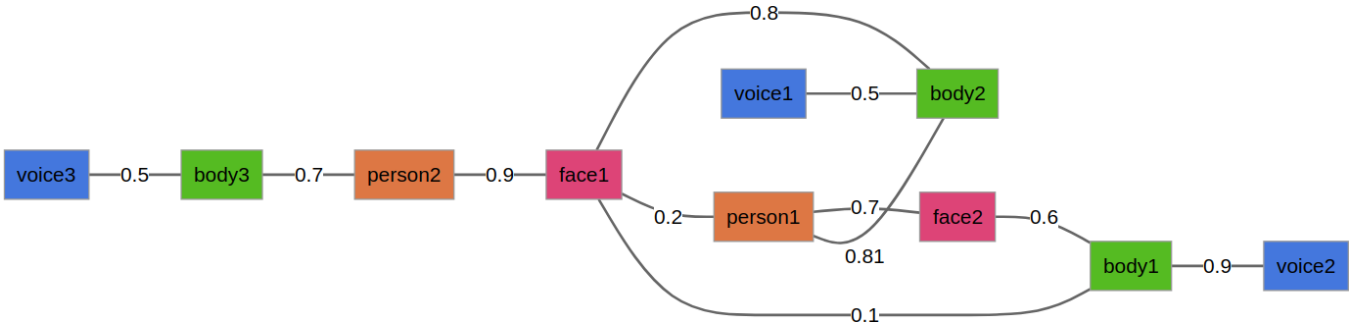

Figure 1: Example of relations graph generated by the robot's perception modules. Numerical values are likelihoods that two features are indeed belonging to the same person.

Note that the values used in Figure 1 were not actually measured; they were chosen for illustrative purpose, also stressing some edge cases of the algorithm.

Looking at this diagram, it seems clear that `person2` should be associated to `face1` and `person1` to `face2`. The associations of bodies to persons is less obvious: `body3` could be associated to `person3` (direct relation, likelihood $\mathcal{L} = 0.7$), but `body2` seems to be likely to 'belong' to `face1` ($\mathcal{L} = 0.8$): if `face1` is already associated to `person2`, then maybe should `body2` also be associated to the same person? Similar ambiguities can be found for `person1`: is it more likely to be associated to `body2` or `body1` (via `face2`)?

A first naive approach to the problem involves looking for the most likely path between a person and each of its features (i.e. pathfinding with weights = 1 - likelihood). However, this approach does not work well in practice, due to some of the unique constraints of the problem (like the fact that an association may contain at most exactly one feature of each type).

Our algorithm instead search all possible *partitions* of the initial graph, and select the one that minimizes the number of associations, while maximizing *affinity*, i.e. the sum of likelihoods of each associations. In practice, our algorithm yields the three associations presented in Figure 2 (assuming in this example a likelihood threshold of 0.4).

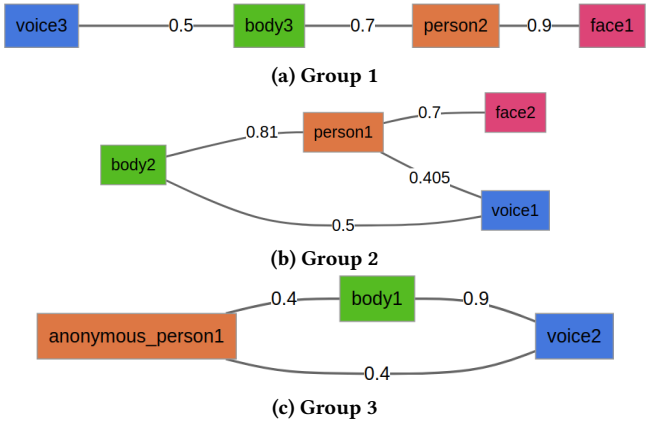

Figure 2: Partition of the graph in Fig. 1 that maximise overall likelihood. One anonymous person had to be created in Group 3 for the features `body1` and `voice2`, as they would otherwise be dangling. Note that the `person1`–`voice1` edge in Group 2 does not exist in the original data: it is a *computed edge*, used to ensure stability if `body2` is not detected anymore and disappear.

3 DEFINITIONS

DEFINITION 1: a *feature* is a face, body, or voice. Accordingly, the *feature type* is one of *face*, *body* or *voice*. (*Note: these 3 features happen to be ones currently specified in the ROS4HRI specification. Additional features could be added – nothing in the algorithm is specific or limited to these three ones.*)

DEFINITION 2: a *node* is either a feature or a person, and accordingly *node types* are one of face, body, voice or person.

DEFINITION 3: *nodes* (e.g. either persons or features) might be *connected* to each other if a likelihood of association has been published between them (specified in ROS4HRI as a message over the `/humans/candidate_matches` topic). Together they span a graph called the *persons–features graph*.

DEFINITION 4: A *permissible path* between two nodes is a path that does not contain more than one node of each type (including the start and end nodes). The full path likelihood is computed as the product of likelihoods along the path.

DEFINITION 5: A node may be *associated* to another one if:

- it is not already associated to a node of the same type;
- there exists a permissible path between the two nodes;
- the likelihood of the resulting path is above the *likelihood threshold*.

DEFINITION 6: an *association* is a connected set of nodes, such that each node in the set is *associated* to at least one other node. As such, *association* are connected subgraphs of the original persons-features graph.

DEFINITION 7 an association's *affinity* is the *sum* of all the likelihoods on the minimum spanning tree (computed using weights equal to (1 - likelihood)) of the association's graph.

4 ALGORITHM REQUIREMENTS

R1: maximum likelihood: associations between persons and their features must be chosen so that they maximize the total *affinity* of all associations.

R2: existence of persons: at anytime, every feature must be associated to exactly one person (anonymous or not).
R3: anonymous persons: as a consequence of **R2**, features that are not associated to a person must be associated to temporary 'anonymous' persons. This should still respect the requirements of DEFINITION 5: if two features are associated (e.g. a face and a body), they must be associated to the same anonymous person.
R4: stability: associations with indirect paths (e.g. `A` associated to `C` because `A` connected to `B` and `B` connected to `C`) should be maintained in case of the path is broken (e.g. `B` disappears) *if and only if* the likelihood of the association between the two nodes is superior to the likelihood threshold when the path is broken. In that case, a new *direct* association must be created and the likelihood of that association will remain constant until the association is removed (because one of the two nodes disappeared).

5 ALGORITHM

5.1 The *Mr. Potato* algorithm

The overall aim of the algorithm is to find, for each person in the scene, its most likely *association*. For a given persons-features graph $G(V, E)$ with V the nodes (persons or features), and E the relations between the nodes, with the likelihoods $\mathcal{L}_e$, we want to optimize for the overall likelihood of each of the persons' associations.

ALGORITHM 1: `computeAssociations`; $G(V, E)$ denotes a *persons–features* graph (with nodes V and edges E with likelihood $\mathcal{L}_e$); $p \in \mathcal{P}_G$ denotes a partition of such a graph.

input : $G(V, E)$; likelihood threshold t
output : a partition p of $G(V, E)$

```
// prune edges whose likelihood is below t
foreach e ∈ E do
    if L_e < t then
        remove(G, e)
// remove all existing anonymous persons
foreach v ∈ V do
    if v is anonymous person then
        remove(G, v)
p_final = ∅
foreach G_c in connectedComponents(G) do
    P ← getPartitions(G_c)
    // only keep the most compact partitions; ie
       partitions with minimum of subgraphs
    P_compact ← {p̃ ∈ P; len(p̃) = min_{p∈P}(len(p))}
    // select partition with highest affinity
    p_best = argmax_{p∈P_compact}(partitionAffinity(p))
    foreach G_best ∈ p_best do
        // if no person in this subgraph, insert a new
           anonymous person, with likelihood = t
        addAnonymousPersons(G_best)
        fullyConnectPersons (G_best)
        p_final = p_final ∪ {G_best}
return p_final
```

To this end, our algorithm takes the following steps:

(1) Compute all possible partitions of the graph G, with the constraints that (1) each subgraph can not contain more than one node of each type (e.g., two faces, or two bodies is not permissible); (2) the subgraphs are connected. Each subgraph is a candidate person association;
(2) Compute how 'good' each partition is, and select the best one. How 'good' is measured by (1) having a minimal number of subgraphs; (2) having the highest total affinity (computed as the sum of affinities of each subgraph in the partition);
(3) Add *anonymous persons* to subgraphs that do not already have a person (because, for instance, a face is detected, but not yet recognised);
(4) If necessary, create *computed* edges between the person and each of its features (so that the association is stable in case of one feature disappearing).

Algorithm 1 implements the whole process, with Algorithm 2 specifically implementing Step 1; Algorithm 3 implementing Step 2; and Algorithm 4 implementing Step 4.

ALGORITHM 2: `getPartitions` *(same notations as Alg. 1)*

input : $G(V, E)$; likelihood threshold t
output : a set of partitions

```
// Recursively computes and returns a set of partitions
   that have at most one node of each type
func buildPartitions V
    head = V[0]
    if len(V) = 1 then
        return {{{head}}} // one partition with only one
           graph containing only one node
    P = buildPartitions(V[1:...])
    P' = ∅
    foreach p ∈ P do
        foreach G ∈ p do
            if type(head) ∉ G then
                p' = {G'; G' ∈ p; G' ≠ G} ∪ {G ∪ {head}}
                P' = P' ∪ {p'}
        P' = P' ∪ {p ∪ {{head}}}
    return P'
func getPartitions G(V,E)
    P ← buildPartitions(V)
    // only keep partitions where each sub-graph is fully
       connected
    P_c = ∅
    foreach p ∈ P do
        foreach G ∈ p do
            if not isFullyConnected(G) then
                continue with next p
        P_c = P_c ⋃ p
    return P_c
```

ALGORITHM 3: `partitionAffinity` *(same notations as Alg. 1)*

```
affinity = 0
foreach G in partition do
    T ← minimumSpanningTree(G, weights = (L ↦ 1 − L))
    affinity += Σ_{e∈T} L_e
return affinity
```

ALGORITHM 4: `fullyConnectPersons` *(same notations as Alg. 1)*

input : $G(V, E)$, containing at most one node per node type
input : likelihood threshold t
output : $G(V, E)$ with fully connected persons

$v_p = v \in V, \text{type}(v) = \text{person}$
$E_{\text{prev\_computed}} = \emptyset$
$E_{\text{newly\_computed}} = \emptyset$
foreach $e_p \in E$; v_p *connected to* e_p **do**
  if *isComputedEdge(e)* **then**
    $E_{\text{prev\_computed}} = E_{\text{prev\_computed}} \cup \{e_p\}$
    remove(G, e_p)
foreach $v \in V, v \neq v_p$ **do**
  if ***not*** v *directly connected to* v_p **then**
    $\mathcal{L}_{v,v_p} = \text{shortestPath}(G, v_p, v, \text{weights} = (\mathcal{L} \mapsto log(\frac{1}{\mathcal{L}})))$
    if $\mathcal{L}_{v,v_p} > t$ **then**
      $e_{v,v_p} = \{v \rightarrow v_p; \mathcal{L} = \mathcal{L}_{v,v_p}; isComputed = true\}$
      $E_{\text{newly\_computed}} = E_{\text{newly\_computed}} \cup \{e_{v,v_p}\}$
      add(G, e_{v,v_p})
foreach $e_{prev} \in E_{prev\_computed} \setminus E_{newly\_computed}$ **do**
  add(G, e_{prev})
return G

5.2 Step-by-step example

We can apply Algorithm 1 to the graph pictured in Diagram 1, using a likelihood threshold of $t = 0.4$.

We first prune the two relations whose likelihood $\mathcal{L}$ is below the threshold ($e_{face1,person1}$ and $e_{face1,body1}$).

Then, `getPartitions` (Alg. 2) generates 352 possible partitions from the remaining graph, which reduces to $|\mathcal{P}_{compact}| = 6$ partitions of minimal size $n = 3$ subgraphs.

These 6 partitions have affinities ranging from 5.01 to 4.7, the best partition p_{best} being pictured in Figure 2. The affinity value of 5.01 can easily be computed from Figure 2 by summing likelihoods across all associations, omitting both the link between `person1` and `voice1` in Group 2 (as this is a computed edge that does not yet exist in the graph at this point in the algorithm); and the links to `anonymous_person1` in Group 3, as anonymous persons are similarly added at the later point in the algorithm.

The next step in the algorithm is to create and add anonymous persons for associations that lacks a person, like Group 3 in Figure 2. In the actual implementation (see below), the ID of the anonymous person is generated from a hash of the connected features' IDs, ensuring the same anonymous person ID will be reused if the features of the association are the same.

Next, we compute direct edges between persons and each of their features that are not already directly connected. This is visible for instance in Group 2: a new link `person1`–`voice1` is created, with likelihood $\mathcal{L}_{\text{person1, voice1}} = \mathcal{L}_{\text{person1, body2}} \times \mathcal{L}_{\text{body2, voice2}} = 0.81 \times 0.5 = 0.405$, corresponding to the likelihood along the maximum-likelihood path (i.e. the shortest path computed with weights = $log(1/\mathcal{L})$) between the person and the feature. If `body2` is removed from the graph (for instance, the body is occluded, and not detected anymore), adding this computed edge ensure that the association between the person and the voice is maintained. Note that if the computed likelihood would have been below the threshold, we would not have added this new edge.

The algorithm concludes by returning the three computed associations (Figure 2) that maximize the overall affinity.

5.3 Complexity

We note n the number of nodes in the largest connected component of the persons–features graph. The time complexity of the `computeAssociations` algorithm is dominated by the recursive `getPartitions` algorithm. The complexity of `getPartitions` can be computed to be $O(n^n)$. Indeed, one association is made of, at most, one node of each type. As such, there is $\propto n^4$ ways of creating a valid association. A partition is made of at most n associations (in the worst case of associations with a single node in each of them). Therefore, to build a partition, we need to pick at most n elements amongst n^4. As such, the complexity of finding all permissible partitions is $O(C(n^4, n)) = O((n^4)^n) = O(n^n)$.

While the complexity exponentially increases with the number of nodes, in practice, n tends to remain small (10), as it is limited by the number of people that can be seen by the robot's camera at a given time.

6 IMPLEMENTATION

We provide a C++ implementation of the algorithm as an open-source ROS node (supports both ROS 1 and ROS 2; source code: https://github.com/ros4hri/hri_person_manager).

The node implemented the ROS4HRI REP-155 standard, and, as such, listens for likelihoods of associations between persons and features on the `/humans/candidate_matches` topic. The likelihood threshold can be configured via the ROS parameter `/humans/match_threshold`.

Our implementation relies on the `boost::graph` library to efficiently represent the persons-features graph. Connected components are computed using `boost`'s `connected_components` algorithm that uses a Depth-First-Search approach; likewise, minimum spanning trees are calculated using the `boost` implementation of the Kruskal's algorithm; and shortest paths between nodes are computed using `boost` implementation of the Dijkstra algorithm.

The current implementation is not multi-threaded; it runs on the graph pictured in Figure 1 in about 55ms on an Intel i7 11th CPU. As expected, the execution time is almost entirely dominated by the generation of partitions (`getPartitions`), which is itself largely dominated by memory allocation calls. As such, further optimizations are possible.

7 CONCLUSION

We present in this paper a novel algorithm to perform probabilistic fusion of human 'features' (like faces, bodies, voices) into complete models of persons. We also introduce an open-source C++ implementation of the algorithm, fully compatible with the ROS4HRI REP-155 standard. This implementation is deployed and routinely used on our social robots.

ACKNOWLEDGMENTS

This work has been funded by the EU H2020 SPRING project (grant agreement No.871245). Additional funding has been received from the EU H2020 ACCIO TecnioSpring INDUSTRY (grant agreement no. 801342, TALBOT project).

REFERENCES

[1] Y. Mohamed and S. Lemaignan. 2021. ROS for Human-Robot Interaction. In *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. https://doi.org/10.1109/IROS51168.2021.9636816