Complex Intell. Syst. (2016) 2:45–59
DOI 10.1007/s40747-016-0014-8

ORIGINAL ARTICLE

Interoperable multi-agent framework for unmanned aerial/ground vehicles: towards robot autonomy

Willson Amalraj Arokiasami[1] · Prahlad Vadakkepat[1] · Kay Chen Tan[1] · Dipti Srinivasan[1]

Received: 18 November 2015 / Accepted: 2 April 2016 / Published online: 28 April 2016

Abstract Multi-agent architectures for autonomous robots are generally mission and platform oriented. Autonomous robots are commonly employed in patrolling, surveillance, search and rescue and human-hazardous missions. Irrespective of the differences in unmanned aerial and ground robots, the algorithms for obstacle detection and avoidance, path planning and path-tracking can be generalized. Service-oriented interoperable framework for robot autonomy (SOIFRA) proposed in this paper is an interoperable multi-agent framework focusing on generalizing platform-independent algorithms for unmanned aerial and ground vehicles. As obstacle detection and avoidance are standard requirements for autonomous robot operation, platform-independent collision avoidance algorithms are incorporated into SOIFRA. SOIFRA is behaviour based and is interoperable across unmanned aerial and ground vehicles. Obstacle detection and avoidance are performed utilizing computer vision-based algorithms, as these are generally platform independent. Obstacle detection is achieved utilizing Hough transform, Canny contour and Lucas–Kanade sparse optical flow algorithm. Collision avoidance performed utilizing optical flow-based and expansion of object-based time-to-contact demonstrates SOIFRA's modularity. Experiments performed, utilizing TurtleBot, Clearpath Robotics Husky, AR Drone and Hector-quadrotor, establish SOIFRA's interoperability across several robotic platforms.

This work was supported by the Singapore Ministry of Education Academic Research Fund Tier 1 under the Project R-263-000-A12-112.

✉ Willson Amalraj Arokiasami
willson@u.nus.edu

1 National University of Singapore, 4 Engineering Drive 3, Singapore 117576, Singapore

Keywords Behaviour based multi-agent framework · Collision avoidance · Unmanned ground vehicle · Unmanned aerial vehicle

Introduction

Unmanned autonomous vehicles are robots, capable of intelligent actions and motions, operating without a guide or teleoperator. Unmanned vehicles are aiding or replacing humans in various tasks such as space and undersea exploration [10,19], remote repair and maintenance [7], remote sensing [35], disaster management [25] and military reconnaissance [6]. The dynamic nature of autonomous robot's operating environments demand robots with rapid online decision-making ability, fault tolerance and scalability. Agent-oriented approaches, suitable for designing unmanned robot systems, break down sequential top-down programs into a set of simple, distributed and decentralized processes that have direct access to sensors and actuators of the robot [23]. An agent is a computational system that tries to fulfil a set of goals in a complex and dynamic environment [20,43]. Groups of several agents working cooperatively or non-cooperatively to solve tasks form a multi-agent system [28,41]. The inherent modular and distributed nature of the mult-agent system offers scalability, fault tolerance and parallelism [21,36]. Behaviour-based multi-agent architectures improve the rapid online decision-making ability of a robot by decomposing a system into subsystems with task-achieving behaviours [22,39]. Multi-agent architectures for service-oriented unmanned vehicle applications are generally platform oriented, leading to varied architectures for the same applications [13–15,31,33,40]. Algorithms for service-oriented unmanned vehicle applications are generalized and it is possible to use these algorithms on various platforms irrespective of the details of their implementations. The proposed service-oriented interoperable framework for robot autonomy (SOIFRA) provides a framework for algorithms that can be generalized and non-platform specific. Intelligence and ability to detect and avoid obstacles are basic requirements for autonomous operation of unmanned vehicles. In this work, SOIFRA provides a framework that includes non-platform-specific collision avoidance algorithms.

Coupled layered architecture for robotic autonomy (CLARAty) is a framework for heterogeneous robot platforms with generic and reusable robotic components [27]. CLARAty provides a framework for generalized algorithms applied to rover platforms irrespective of the implementation details. Agent design patterns defining common features allow introduction of new hardware and software components without modifying the architecture [3]. The target-drives-means (TDM), behaviour-based interoperable software framework for humanoid robots, supports flexible behaviours [4]. The hybrid deliberative/reactive architecture (HDRC3) is a distributed architecture for unmanned aircraft systems [9]. In HDRC3, the essential generic functionalities of a UAV are isolated for effective integration of low-level (navigational subsystem, low-level control with motion planning) and high-level (mission planning and execution) functionalities. The intelligent control architecture (ICA) [16] is a generic capability-oriented architecture for autonomous marine robots. ICA enables multiple collaborating marine vehicles to autonomously carry out underwater intervention missions. The proposed work SOIFRA is a behaviour-based multi-agent framework for autonomous unmanned aerial and ground vehicles.

Collision avoidance, which is crucial in mobile robot navigation, comprises obstacle detection and avoidance [42]. Vision-based obstacle detection is powerful and popular in unmanned aerial and ground vehicles [18,44]. Vision sensors, in close proximity, provide detailed information about an environment. Appearance-based and optical-flow-based techniques are commonly used in mapless vision-based navigation. Appearance-based methods rely on basic image processing techniques to differentiate the obstacles from the background. Optical-flow-based techniques utilize apparent motion of objects in scenes [26]. Optical flow, a biologically inspired approach for obstacle avoidance, is the pattern of apparent motion of pixels in successive image frames [34]. The time derivative of positions of image points in an image plane define a motion field. Temporal change in an image sequence constitutes the optical flow field which approximates the motion field. Optical flow can be computed using spatio-temporal intensity derivatives (differential method), feature matching techniques (correlation approach) and velocity-tuned filters (frequency-based method) [44]. Lucas–Kanade method, a differential method for optical flow estimation, is useful for sparse optical flow computation [24]. For larger pixel motions, a pyramidal approach for Lucas–Kanade method is suitable, where the standard algorithm is applied recursively to re-sized versions of the image. Obstacle avoidance for autonomous robots based on optical flow from a monocular camera is actively researched upon [8,12,29,30,32,44]. In most cases, the translational part of the optical flow field is utilized to estimate the distance to the obstacle based on which obstacle avoidance is performed.

Obstacle avoidance algorithms generate motion instructions combining distance to the obstacle(s) and vehicle motion information. Range sensors are often utilized to obtain distance information from environment. Recent advancements have enabled computer vision algorithms to estimate depth information utilizing multiple image frames. Obstacle avoidance algorithms are of two categories: global and local. Global approaches compute optimal robot trajectory, off-line, utilizing a complete model of the robot's environment, while computationally efficient local or reactive approaches generate sub-optimal robot trajectories. Local approaches such as potential field method (PFM), virtual force field (VFF), vector field histogram (VFH) and dynamic window approach (DWA) are fast and computationally efficient, as only a small subset of obstacles close to the robot is considered. In potential field method [17], a robot is subjected to attractive force from the target and repulsive forces from obstacles. Virtual force field and vector field histogram are extensions of PFM [38]. In the dynamic window approach, robot dynamics is considered to compute admissible velocities for safe robot motions [11]. Stereo vision is a common technique for obtaining 3D depth information from 2D planar images [1]. Two obstacle avoidance algorithms, one analyzing optical flow fields from monocular 2D images to obtain obstacle(s) distance information and the other utilizing expansion of an object to obtain distance information, are utilized in the proposed framework.

Research contributions

SOIFRA, an interoperable multi-agent framework for unmanned aerial and ground vehicles proposed in this work, provides a framework for generalized and platform-independent algorithms. As collision avoidance is standard for autonomous operation of unmanned robots, algorithms for obstacle detection and avoidance are incorporated into SOIFRA. For obstacle detection, the Lucas–Kanade sparse optical flow algorithm, the Hough transform and the Canny contour mapping algorithm are utilized. Two obstacle avoidance methods, an optical flow-based method and expansion-of-obstacle-based method, are utilized to demonstrate the modular nature of the framework proposed. A case study is presented where a robot is expected to navigate an unknown area autonomously avoiding obstacles in its path. Error performances of the obstacle avoidance methods are studied.

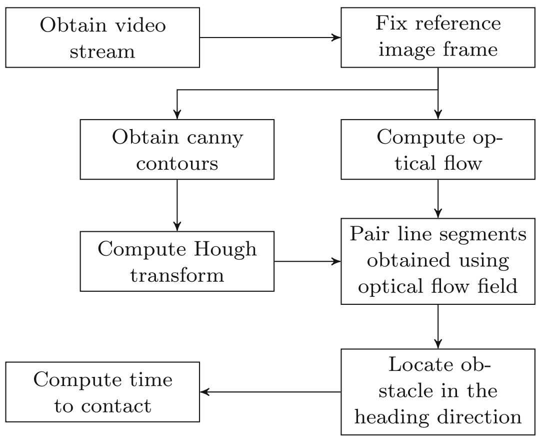

Obtain video stream
Fix reference image frame
Obtain canny contours
Compute optical flow
Compute Hough transform
Pair line segments obtained using optical flow field
Locate obstacle in the heading direction
Compute time to contact

Fig. 1 Process flow for obstacle detection

The framework is tested on Parrot AR Drone, Hector Drone, Clearpath Robotics Huskey and Turtlebot, in simulation and Parrot AR Drone and TurtleBot in real-time environments. The experiments illustrate the feasibility in utilizing the same collision avoidance algorithm for autonomous unmanned aerial and ground vehicles, performing identical tasks.

The paper is organized as follows. Sect. "Obstacle detection" explains the obstacle detection proposed. Section "Obstacle avoidance" elaborates on the obstacle avoidance algorithm and the methods to estimate distance to the obstacle detected. An overview of the multi-agent framework proposed is explained in Sect. "Overview of SOIFRA". Simulation results for the case study utilizing Clearpath Huskey, Turtlebot, Hector-drone and AR drone are presented in Sect. "Simulation results". Experimental results for the case study utilizing Turtlebot and AR drone are presented in Sect. "Experimental results". Conclusions and future directions of the work are outlined in Sect. "Conclusion and future works".

Collision avoidance

Collision avoidance is the ability of the robot to detect and avoid obstacles along its path. Obstacle detection is performed by applying Canny contours, Hough transform and optical flow. Once it is established that the robot is in collision course with the obstacle identified, the obstacle is tracked. The algorithms to detect obstacle is explained in this section.

Obstacle detection

Obstacle detection is the first step in collision avoidance. The process flow for static obstacle detection is shown in Fig. 1. The simulation and real-time environment, where the robots operate, are shown in Fig. 3a and b, respectively. Canny contour, Hough transform and optical flow vectors are utilized to detect obstacles. Contours in the image are identified utilizing the Canny contour algorithm, and line segments in

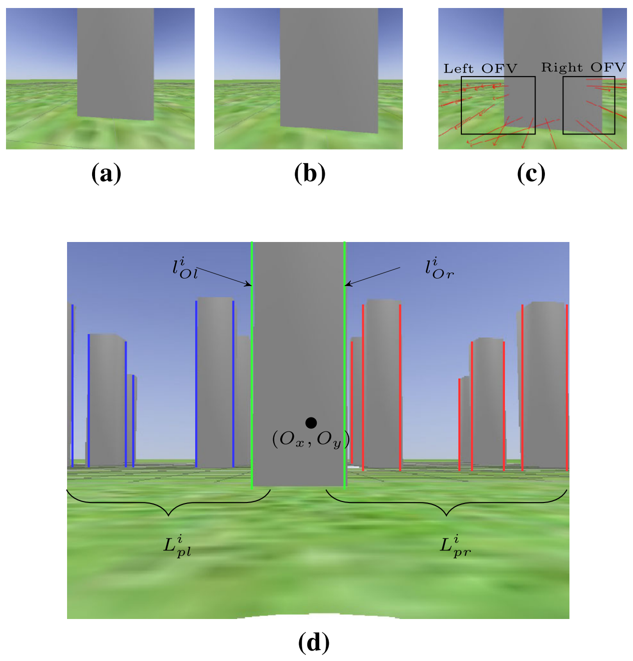


Fig. 2 **c** Shows the optical flow vector generated from an obstacle as the robot moves towards the obstacle and *left* and *right* optical flow vectors generated from *left side* and *right side* of the obstacle. **a** and **b** show the image frames utilized to generate the optical flow vectors shown in **c**. **d** Is an illustration showing the line segments l_i^j obtained for an image i. **d** line segments l^i_j, obtained for an image i, are shown in *red*, *blue* and *green colors*. L^i_{pl} and L^i_{pr} include the *line* segments that are to the *left* (*blue*) and *right* (*red*) of the optical center. l^i_{Ol} and l^i_{Ol} are the line segments (*green*) of L^i_{pl} and L^i_{pr}, that are the closest to the optical center. Only *line* segments with $\alpha^i_j + \theta_t = 90°$ are shown

the image are isolated using Hough transform. The obstacle detection algorithm (Algorithm 1) requires the optical center of the camera (o_x, o_y) and image i at time t as its input. Let L^i be a set of n line segments l^i_j, obtained through Hough transform for image i as shown in Fig. 2d (l^i_j is the jth line segment obtained for image i). α^i_j is the angle associated with a line segment l^i_j and θ_t is the roll angle of the robot at time t of image i (θ_t is zero for ground vehicles). The vertical edges of the objects in an image frame i are split into L^i_{pr} and L^i_{pl} based on its location with respect to the optical centre. If l^i_j is to the left of the optical centre, it is grouped into L^i_{pl} and L^i_{pr} if it is to the right. l^i_{Ol} and l^i_{Or} are the line segments l^i_j of L^i_{pl} and L^i_{pr}, which are the closest to the optical center. OF^i_{Ol} and OF^i_{Or} are the optical flow vectors generated due to the motion of l^i_{Ol} and l^i_{Or} with respect to time. Lucas–Kanade Sparse optical flow algorithm is utilized for obtaining the optical flow vectors. The obstacle to be avoided is identified utilizing the concept that optical flow vectors from edges of an object do not intersect if the robot is in collision course with the obstacle (Fig. 4). If optical flow

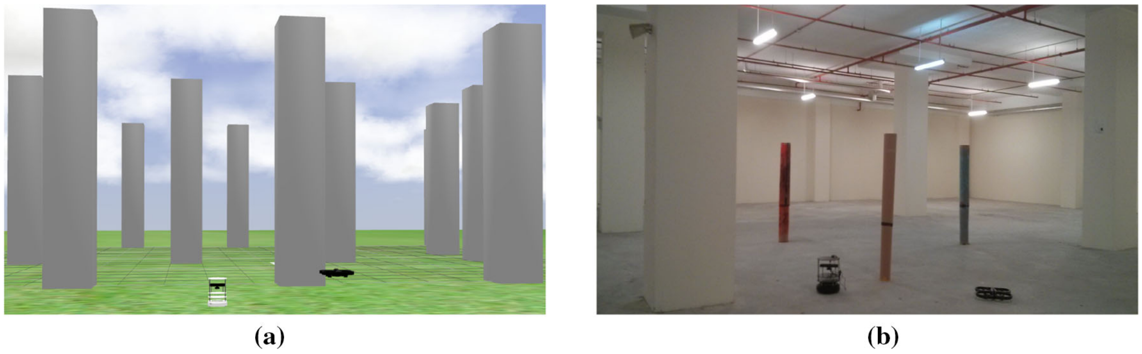


Fig. 3 Image of simulation and real-time operational environments with multiple static obstacles. **a** Simulation environment, **b** real-time environment with Turtlebot and AR Drone

vectors in $OF^i_{\max}$ and $OF^i_{\min}$ do not intersect, then $l^i_{\max}$ and $l^i_{\min}$ are identified as the edges of the obstacle to be avoided. The obstacle detection method proposed is more suitable for detecting structured obstacles that may have vertical edges; for example, walls, pillars and tables.

Algorithm 1 Algorithm for obstacle detection

```
Input: Optical centre of the camera $O(O_x, O_y)$, image $i$ at time $t$
1:  $L^i = \{l^i_j, \alpha^i_j\} = \{\{L_p\}, \{L_e\}\}$   $j = 1, ..n$, (Obtain all the line segments $l^i_j$ and the angle associated with the line segments $\alpha^i_j$ in the image $i$ using Hough transform)
2:  if $\alpha^i_j + \theta_t = 90°$ then
3:      if $l^i_{jx} < o_x$ then
4:          $L^i_{pl} = \{l^i_j, \alpha^i_j\}$
5:      else
6:          $L^i_{pr} = \{l^i_j, \alpha^i_j\}$
7:      end if
8:      $l^i_{Ol}$ is the $l^i_j$ of $L^i_{pl}$ closest to optical center
9:      $l^i_{Or}$ is the $l^i_j$ of $L^i_{pr}$ closest to optical center
10:     $OF^i_{Ol}$ and $OF^i_{Or}$ are the optical vectors that are generated due to change in positions of $l^i_{Ol}$ and $l^i_{Or}$ w.r.t. time.
11:     if Optical flow vectors in $OF^i_{Ol}$ and $OF^i_{Or}$ do not intersect then
12:         Obstacle is detected between $l^i_{Ol}$ and $l^i_{Or}$.
13:     else
14:         Obstacle is not detected.
15:     end if
16: else
17:     $L^i_e = \{l^i_j, \alpha^i_j\}$
18: end if
```

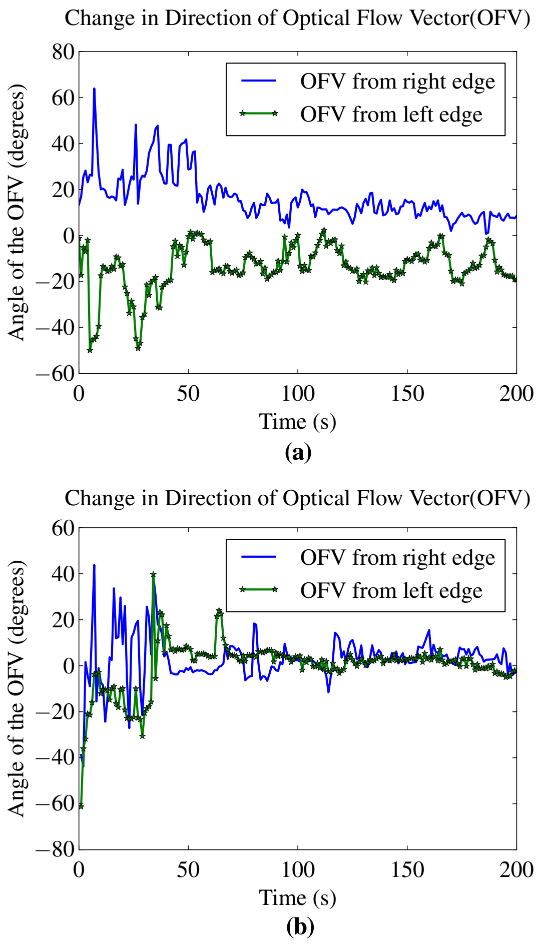


Fig. 4 Change in direction of optical flow vectors, associated with *left edge* (l^i_{Ol}) and *right edge* (l^i_{Or}) of the obstacle while a robot is moving forward. The optical flow vectors from the *left* and *right edges* do not intersect if the robot is in a collision course with the obstacle. **a** Change in direction of the optical flow vectors identified from the *left* and *right* edges of an obstacle in collision path of the robot, **b** change in direction of the optical flow vectors identified from the *left* and *right* edges, when the robot is not in collision course with any obstacle

Obstacle avoidance

Obstacle avoidance is the second phase of collision avoidance. Once it is determined that the robot is in a collision course with an obstacle detected, the distance to that obstacle from the robot is estimated continuously. Distance to the obstacle estimated based on optical flow and expansion of objects are explained in this Section.

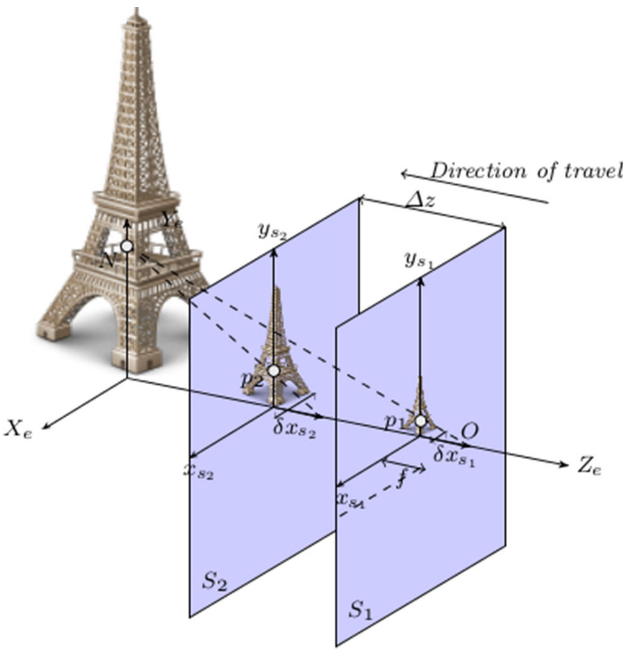


Fig. 5 Projections of a point P onto image planes S_1 and S_2

Estimation of time-to-contact

Time-to-contact (TTC), a quantitative measure, is useful for obstacle avoidance. The 3D information from optical flow fields of 2D images, extracted by time-to-contact, is utilized for obtaining the distance to the obstacle(s). Time-to-contact is estimated utilizing optical flow and expansion of objects-based methods. The error performances of both the methods for the operation environment under consideration are presented in Sect. "Experimental results". Once the time-to-contact is estimated, the distance to the obstacle is obtained by

$$\text{Time-to-contact} = \frac{\text{Distance to the obstacle}}{\text{Velocity of the robot}}. \tag{1}$$

Estimating time-to-contact utilizing optical flow (TTC-OF)

Time-to-contact of an obstacle is determined utilizing the translational component of the optical flow. Time-to-contact estimation is independent of velocity of the robot and distance to the surface of the obstacle [29]. Let f be the focal length of the camera on a robot, facing the direction of motion. For $N(X, Y, Z)$, a point on the obstacle, $p_1(x_{s_1}, y_{s_1}, z_{s_1})$ and $p_2(x_{s_2}, y_{s_2}, z_{s_2})$ are projections on image planes S_1 and S_2, at time instances t_1 and t_2 (Fig. 5). The robot undergoes translation along Z_e with a velocity $V = -\frac{\delta Z}{\delta t}$ over a distance $\Delta z = z_2 - z_1$, approaching the focus of expansion (FOE). From similar triangles,

$$\frac{y_{s_1}}{f} = \frac{Y}{Z} \Rightarrow y_{s_1} = f\frac{Y}{Z}. \tag{2}$$

Differentiating (2) with respect to time provides

$$\frac{\delta y_{s_1}}{\delta t} = f\left(\frac{\frac{\delta Y}{\delta t}}{Z}\right) - fY\left(\frac{\frac{\delta Z}{\delta t}}{Z^2}\right). \tag{3}$$

In (3) $\frac{\delta Y}{\delta t} = 0$, as Y does not change with time. Substituting (2) in (3) and $\frac{\delta Z}{\delta t} = -V$, we get

$$\frac{\delta y_{s_1}}{\delta t} = -y_{s_1}\left(\frac{-V}{Z}\right) \Rightarrow \frac{y_{s_1}}{\frac{\delta y_{s_1}}{\delta t}} = \frac{Z}{V} = \text{TTC}. \tag{4}$$

For a point N, on the obstacle, the distance from its projection, $p1$, on an image plane to the focus of expansion y_{s_1} and length of the optical flow vector $\frac{\delta y_{s_1}}{\delta t}$ are required to estimate the time-to-contact Eq. 4. These calculated optical quantities are adequate in estimating the time-to-contact a point N on the obstacle.

FOE corresponds to the dynamic ambient optical array, which is a single point in space where all the optical flow vectors should emerge. Estimating FOE of an optical flow field is important in calculating the time-to-contact of an obstacle Eq. 4. FOE is estimated by discrete, differential and least-squares-based methods, and performances of these methods are good only when the robot is in pure translation. Theoretically, FOE is the point of intersection of two optical flow vectors. In reality, noise and other errors arising from the steps in computing optical flow vectors affect FOE. In this work, FOE is estimated using least squares solution (Eqs. 5, 6) of all the optical flow vectors identified [37].

$$\text{FOE} = (A^T A)^{-1} A^T b, \tag{5}$$

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{n0} & a_{n1} \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_n \end{bmatrix}, \tag{6}$$

where for each pixel $p_i = (x, y)$ on the image, the associated optical flow vector $V = (u, v)$ gives $a_{i0} = v$, $a_{i1} = u$ and $b_i = xv - yu$.

Estimating time-to-contact utilizing expansion of an obstacle (TTC-EO)

Visual information obtained by monitoring the expansion of an object in visual field is utilized to obtain time-to-contact [5]. If expansion E is defined as the rate of growth of an object in the visual field of a robot, then time-to-contact is given by,

$$\mathrm{TTC} \equiv \frac{1}{E}. \tag{7}$$

In Fig. 5, W is the width of the object, z_{s_2} is the distance between the lens (pinhole) and the object normal to the focal plane at time t_2 and Δz is the distance travelled between t_1 and t_2. δx_{s_1} and δx_{s_2} are the width of the object projected onto image planes S_1 and S_2, respectively. Time-to-contact at t_1 is

$$\mathrm{TTC}_1 = \frac{z_{s_2}}{\Delta z}, \quad \Rightarrow z_{s_2} = \mathrm{TTC}_1 * \Delta z. \tag{8}$$

From the pinhole camera model,

$$\delta x_{s_1} = f\frac{W}{z_{s_2}}, \quad \delta x_{s_2} = f\frac{W}{z_{s_1}} = f\frac{W}{z_{s_2} - \Delta z}. \tag{9}$$

The expansion rate is given by

$$E = \frac{\delta x_{s_2} - \delta x_{s_1}}{\delta x_{s_1}} = \frac{\delta x_{s_2}}{\delta x_{s_1}} - 1. \tag{10}$$

Substitution of (8) and (9) in (10) results in

$$E = \frac{1}{\mathrm{TTC}_1 - 1}. \tag{11}$$

Re-arranging (11),

$$\mathrm{TTC}_1 = \left(1 + \frac{1}{E}\right)(t_2 - t_1), \quad \text{and,} \tag{12}$$

$$\mathrm{TTC}_2 = \frac{1}{E}(t_2 - t_1), \tag{13}$$

where TTC_2 is the time-to-contact at time t_2. Further explanation on implementations of TTC-OF and TTC-EO are in Sect. "Experimental results". Experimental results and error performances of both the methods are shown in Fig. 15 and Table 1 respectively.

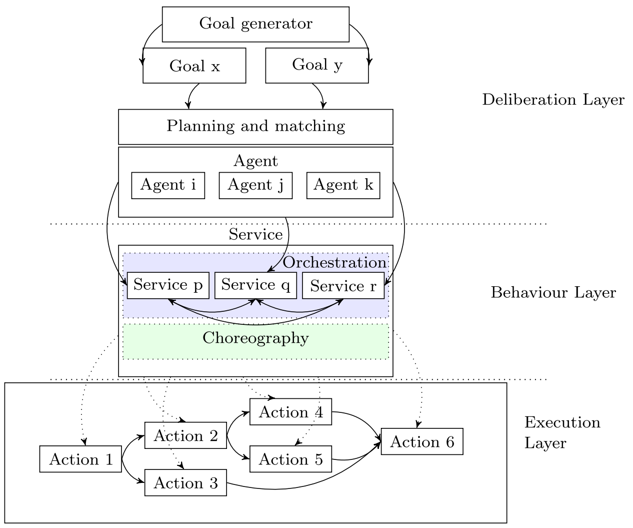


Fig. 6 Architectural overview of the proposed framework (SOIFRA)

Overview of SOIFRA

This section presents the conceptual and structural overview of the multi-agent framework proposed (Fig. 6). The framework is made up of deliberation, behaviour and execution layers. Goal generator, planner–matcher and agents performing various services constitute the deliberation layer. Behaviour layer comprises services, orchestration and choreography of services. Execution layer executes the actions carried out by agent services.

Deliberative layer

The deliberative layer consists of the goal generator, planner–matcher and agents. The goal generator block generates several sub-goals to accomplish a mission. Sub-goals are completed through predefined plans. The mission goal for the case study is to steer a robot towards a target in an unknown environment with obstacles. The goal generator block generates two goals to achieve this mission: a goal to detect and avoid obstacles and a goal to reach the target. Each goal is further divided into sub-goals. For example, a goal to avoid collision is divided into obstacle detection and obstacle avoidance, while the goal to reach a target is divided into a navigation sub-goal. Sub-goals are accomplished through

Table 1 Comparison among mean error, mean absolute error and mean-squared error of distance to the obstacle, computed utilizing TTC-OF and TTC-EO

True distance to the obstacle (m)	TTC-OF (m)			TTC-EO (m)		
	ME	MAE	MSE (m^2)	ME	MAE	MSE (m^2)
4.5–4.0	−3.762	3.762	14.170	0.780	0.800	2.451
4.0–3.5	−3.255	3.255	10.625	0.801	0.801	0.797
3.5–3.0	−2.717	2.717	7.401	0.066	0.080	0.0104
3.0–2.5	−2.194	2.194	4.839	−0.073	0.073	0.063

the plans generated by the planner–matcher module (Sect. “Planner–matcher”). The planner–matcher module helps to achieve a sub-goal by allocating agents to sub-goals based on the services offered by the agents. Agents can collaborate or act independently. Collaborative agents offer services to achieve one or more sub-goals, while non-collaborative agents offer services to achieve only one sub-goal. The steering agent, providing services to achieve the sub-goals, obstacle avoidance and navigation, is a collaborative agent. The obstacle detection agent is a non-collaborative agent that offers services to achieve one sub-goal, obstacle detection. Further explanation on the working of agents and their structure is explained in Sect. “Agent structure”.

Planner–matcher

The planner–matcher module in the deliberation layer (Fig. 6) allocates agents to sub-goals based on the plans generated. Built-in plans are used to generate plans based on the system ontology (Fig. 7). At any point, based on the prevailing states, a competent predefined plan is retrieved from the ontological database and executed. Predefined plans are executed using a sequence of queries as shown below.

- Check if the robot has a predefined plan to complete a goal. If not, notify the planner that the system is not compatible. If it has a predefined plan, then proceed to the next query. The formal query statement associated is shown in Listing 1.
- Check if there are agents that are functionally capable of accomplishing a goal. If not, notify the planner that the robot is functionally not competent. If it is functionally compatible, proceed to the next query. The formal query statement associated is shown in Listing 2.
- Check if the services offered by agents are used by other sub-goals. If they are used by other sub-goals, check if the agents are collaborative. If not, intimate the planner that compatible agents are not available. If they are collaborative, request the agent identified for its services. The formal query statements associated is shown in Listings 3 and 4.

```
SELECT? Goal? Operation? Plan
WHERE {
        system:hasOperation(? Goal?
            ↪ Operation) ^
        system:hasPlan(? Operation)
}
```

Listing 1 Search query to retrieve a pre-defined plan from ontological database

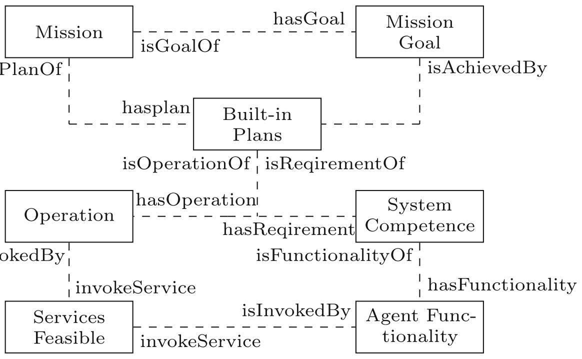


Fig. 7 System ontology

```
SELECT? Agents? Functional_capacity?
    ↪ System_competence
WHERE {
        system:hasFunctionality(? Agent
            ↪ ? Functional_capacity)
}
```

Listing 2 Search query to check the functional competence of the system

```
SELECT? Agents? Services_not_used
WHERE {
        system:hasFunctionality(? Agent
            ↪ ? Functional_capacity)
}
```

Listing 3 Search query to check if the agent services are used other sub-goals

```
SELECT? Agents? Services_used
WHERE {
        system:hasFunctionality(? Agent
            ↪ ? Collaborative_capacity
            ↪ )
}
```

Listing 4 Search query to check if an agent is a collaborative agent

Agent structure

Multi-agent architecture for autonomous robots are of two types: architectures that model an autonomous robot as a single agent and architectures composed of multiple distributed and independent agents to control a robot. The second approach provides modularity and improves fault tolerance of the architecture. The proposed framework follows the second approach where several agents are implemented to achieve a functionally modular agent framework. Autonomous operation of agents in a multi-agent framework requires knowledge about the environment. A world model in an autonomous agent is a representation of knowledge. The world model is made up of internal and external

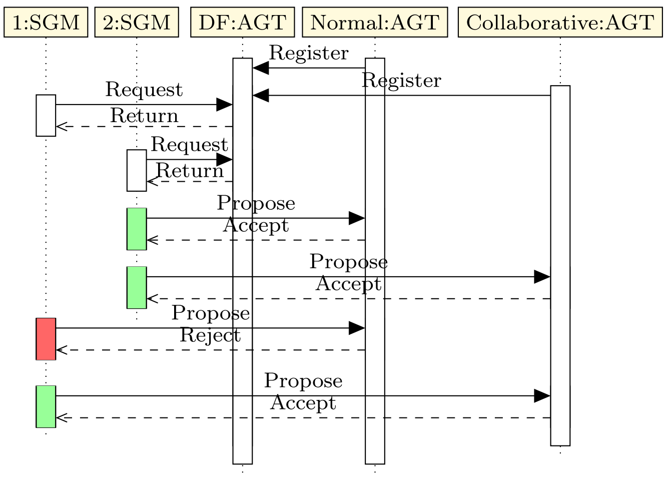


Fig. 8 Illustration of the goal allocation sequence. SGM and AGT refer to sub-goal planner–matcher and agent, respectively. Agents (collaborative and non-collaborative) register with the DF agent. The planner–matcher queries the DF agent to get a list of services offered by the agents. If the planner–matcher requires the service of an agent, it sends a request. The agent accepts or rejects it based on its availability and its collaborative nature

models. The internal model describes the self-knowledge of an agent. Information about the internal operations and services of the agent form the internal model. Knowledge about surroundings and knowledge in social context represent the external model. Interactions among agents and effects of agent's services on the environment form the external model. Information about events in the operating environment of an agent is obtained directly from sensors or agent's services. Internal and external world models form the belief of the agents in the belief–desire–intention model (BDI) -based framework developed. Desire of an agent is represented by the services and actions offered by the agent to carry out the predefined plans allocated by the planner–matcher. The directions of the planner–matcher to achieve a sub-goal represent agent's intentions.

Agents in the proposed framework are implemented using the Java Agent Development Framework (JADE). FIPA protocol is used for communication among agents. Agents register their services with a directory facilitator agent (DF agent), enabling the planner–matcher to allocate agents to sub-goal(s). This framework has collaborative and non-collaborative agents. Collaborative agents offer services that may be utilized to achieve single or multiple sub-goals. Non-collaborative agents achieve only one sub-goal. Sequences of operations by planner–matcher for allocating collaborative and non-collaborative agents to sub-goals are illustrated in Fig. 8. Collaboration among agents is achieved through a priority index. The priority index of a collaborating agent is updated through the parameter server of robot operating system (ROS), utilizing the *dynamic_reconfigure* package. Obstacle avoidance and navigation agents are non-collaborative, achieving their respective sub-goals through collaboration with the steering agent. The steering agent assigns higher priority to the navigation agent when collision is not detected. The priority index of the steering agent changes when a collision is detected and higher priority is assigned to the obstacle avoidance agent. Once the obstacle is avoided, the priority index is updated, resulting in navigation agent attaining higher priority.

Behaviour layer

Services performed by agents and orchestration and choreography of services are a part of the behaviour layer. Services depict the functional capacity of an agent. Services are implemented by combining various actions performed by an agent. Orchestration of services is the process where actions of an agent are combined into a service forming the agent's behaviour. Activities, implemented as *rosnodes*, are combined to form a service. Orchestration of services plays an important role in achieving interoperability. Choreography of services represent communication among agents through messages. Choreography of services is implemented utilizing *rostopics* (named buses over which *rosnodes* exchanges messages).

Execution layer

The execution layer comprises actions performed by agents. Obtaining a video stream, detecting obstacles and computing time-to-contact are the actions performed by the obstacle detection agent. Functional decomposition of actions helps in improving agent modularity. For example, the method to compute time-to-contact can be replaced without affecting other actions of the obstacle detection agent or the structure of the framework. This functionality is demonstrated by utilizing two methods to compute time-to-contact separately for the same mission. Basic actions may be required by more than one agent. For example, obstacle avoidance and navigation agents require the basic actions for controlling the velocity and orientation of the robot. This is achieved through collaboration among agents as explained in Sect. "Agent structure".

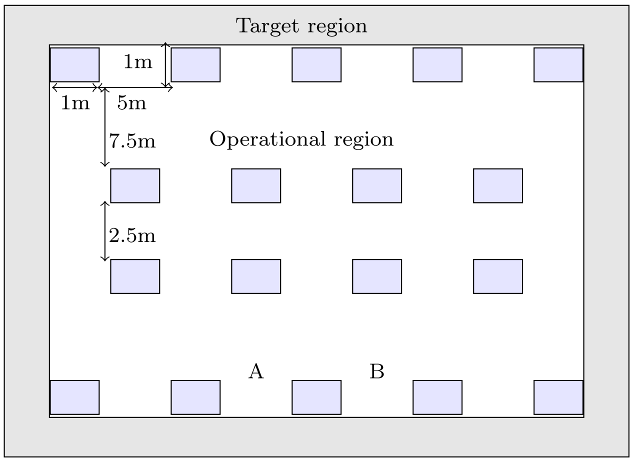


Fig. 9 Layout of the simulation environment (not drawn to scale). The *grey* region indicates the target region and the *white region* indicates the operational region. The mission is completed once the robot reaches the *grey* target region

Results and discussion

This section presents the experimental and simulation results. Both simulation and real-time experiments are performed to demonstrate the interoperability and modularity of SOIFRA in accommodating multiple platform-independent algorithms. The robot's mission is to reach a target destination in an unknown environment while avoiding obstacles utilizing SOIFRA. This mission helps to study the performance of the obstacle detection agent and the collaboration between the steering agent and obstacle avoidance agents. Figure 9 shows the layout of the simulation environment. Figure 3 shows the operational environments for the simulation and real-time experiments. In Fig. 9, the white region corresponds to the robot operation region and the grey region indicates the target region. A and B (Fig. 9) are the robot starting locations. The mission is completed when the robot reaches the target region.

Let Str:AGT and Det:AGT represent the steering agent and the obstacle detection agent; PS:ROS and topic:ROS denote the parameter server and *rostopic*, while Detc:SRV, Video:SRV and TTC:SRV denote the obstacle detection service, video stream service and time-to-contact service, respectively. The obstacle detection agent, upon initiation, starts the actions for video service and obstacle detection service and publishes the control velocity for the robot. The steering agent subscribes to the velocity commands from the obstacle detection agent. When an obstacle is detected, the detection agent initiates the time-to-contact service and updates control velocity for the robot. Two experiments are conducted with time-to-contact estimated utilizing optical flow (TTC-OF) and expansion of the obstacle (TTC-EO) -based methods separately to demonstrate the modularity of SOIFRA. If the time-to-contact service estimates that the distance to the obstacle is less than the critical distance λ (the minimum distance to the obstacle within which action must be taken to avoid an obstacle), the obstacle detection agent updates the ROS parameter server. As a result, the priority index of the steering agent is updated resulting in initiating obstacle avoidance process. If the distance to the obstacle is more than the critical distance, the robot continues to follow its previous path. The critical distance, determined based on the size and linear velocity of the robot, is fixed at 2.5 m for the current mission. Once the obstacle detected is avoided, the steering agent informs the obstacle detection agent that

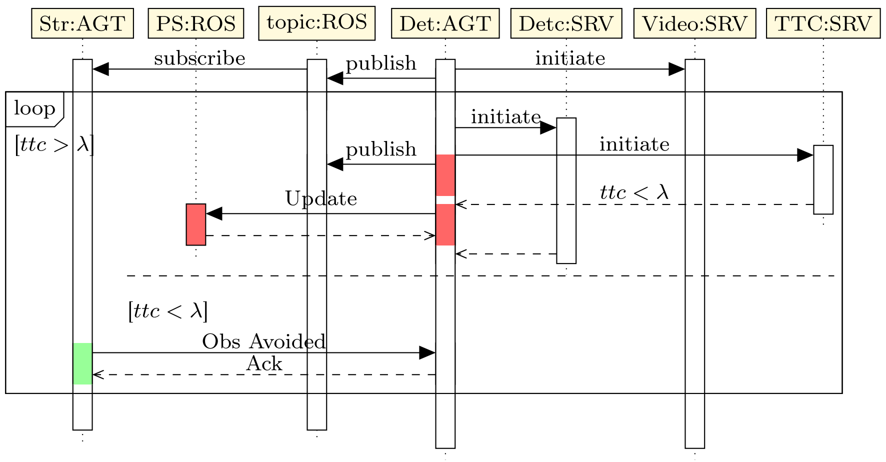


Fig. 10 Operational sequences for obstacle detection and avoidance. AGT and SRV represent agent and service. The detection agent (Det:AGT) obtains video stream utilizing video:SRV service and starts detecting obstacles through the obstacle detection service, Detc:SRV. When an obstacle is detected, Det:AGT initiates TTC:SRV to estimate the distance to the obstacle. When the estimated distance to the obstacle is less than the critical distance λ, Det:AGT updates on the parameter server, PS:ROS. The steering agent (Str:AGT) informs Det:AGT after the obstacle is avoided

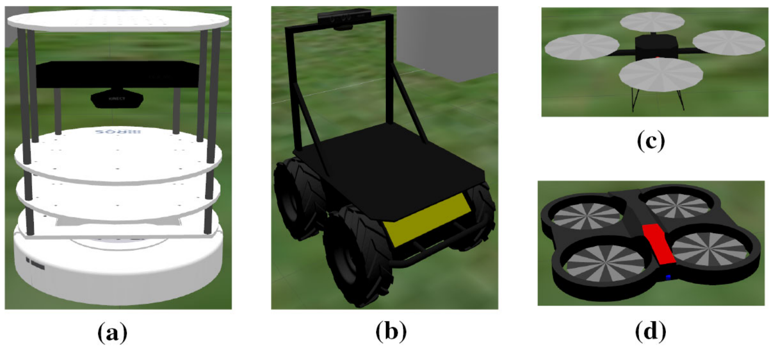


Fig. 11 Visulization of robots in the simulation environment. **a** Turtlebot, **b** Clearpath Husky, **c** Hector-quadrotor, **d** AR Drone

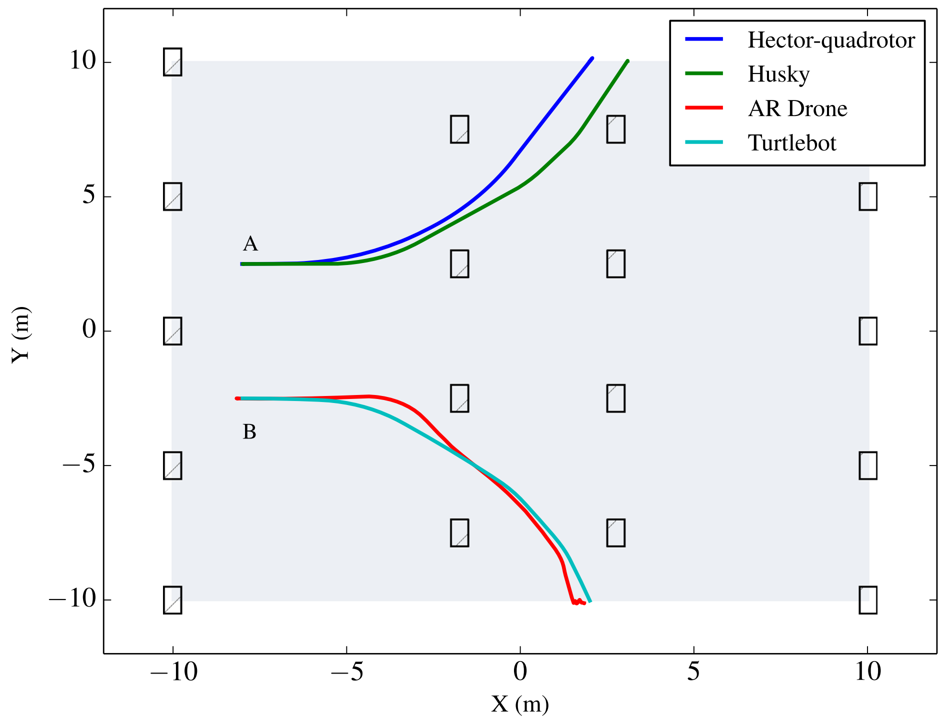


Fig. 12 Simulation of a mission where two ground robots and two aerial robots use TTC-EO for obstacle avoidance. *Top view* (bird's eye view) of the path taken by the robots is shown. Hector-quadrotor and Husky start from *A*, while AR Drone and Turtlebot start from *B*. Each mission is carried out separately

the obstacle is avoided and the navigation agent directs the steering agent to continue to follow a straight path. Figure 10 shows these sequences of operations for obstacle detection and avoidance.

Simulation results

Simulations are carried out utilizing two ground robots (Turtlebot and Clearpath Husky) and two aerial robots (AR Drone and Hector-quadrotor). Fig. 11a–d shows the visualizations of Turtlebot, Clearpath Husky, Hector-quadrotor and AR Drone in the simulation environment. Turtlebot, Clearpath Husky and Hector-quadrotor use the simulated model of Microsoft Kinect as vision sensor, while AR Drone uses a simulated camera. All the cameras produce images with 640 × 480 resolution. *Gazebo*, a 3D simulator for robots, is utilized for simulation. *Gazebo* offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Open dynamics engine (ODE) and open source high-performance library for simulating rigid body dynamics is utilized as the physics engine.

Figures 12 and 13 show the best results for simulations utilizing TTC-EO and TTC-OF obstacle avoidance algorithms, respectively. Each simulation (obstacle avoidance with TTC-OF and TTC-EO) is repeated five times separately. Figures 12 and 13 show the X–Y plane (top view) of the simulation environment. The operating region is a 20 × 20 m square, as indicated in Fig. 9. The robots start from starting locations A or B. Clearpath Husky and Hector-quadrotor start from A and Turtlebot and AR Drone start from B, utilizing TTC-EO. The positions interchange while TTC-OF is utilzed for obstacle avoidance (Clearpath Husky and Hector-quadrotor start from B and Turtlebot and AR Drone start from A). If the robots move out of the operating region (indicated by the grey shade), the mission is complete. The ground robots move with the same velocity and the aerial robots move with the same velocity. The aerial robots move at a higher velocity

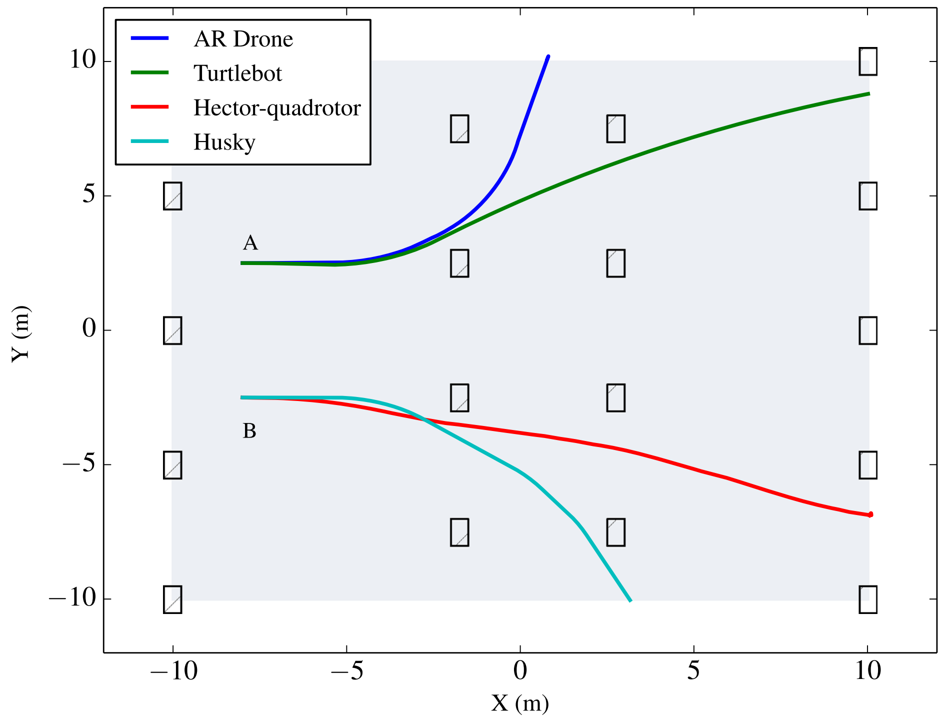


Fig. 13 Simulation of a mission where two ground robots and two aerial robots use TTC-OF for obstacle avoidance. *Top view* (bird's eye view) of the path taken by the robots is shown. AR Drone and Turtlebot start from *A*, while Hector-quadrotor and Husky start from *B*. Each mission is carried out separately

compared to the ground robots. The robots move forward in a straight line path if there is no obstacle detected or if the distance to the obstacle is greater than the critical distance or if the obstacle avoidance process is completed.

Figure 12 shows that all the robots avoid two obstacles before they exit the operational region. There are differences in the path taken by the ground and aerial robots, though they follow the same direction. This is due to the inherent differences between a ground robot and an aerial robot. Ground robots are more stable and can turn in a stabilized manner, while the aerial robots need some time for stabilization after a turning manoeuvre. Once the distance to the obstacle detected is less than the critical distance, the robots perform a turning manoeuvre to avoid the obstacle. If the robot has moved 0.8 m from the point where a turning manoeuvre is initiated and if there is no obstacle detected, the robot stops the turning manoeuvre and moves forward in the same direction. The direction of the turn depends on the position of the obstacle in the robot's path. If the robot senses that obstacle is located to its left, a right turn is performed and vice versa. This is the reason for different turning directions when the robots start from A and B, though the operational environment is symmetrical.

Figure 13 shows the simulation results when TTC-OF is utilized for obstacle avoidance. Turtlebot and AR Drone start from A and Clearpath Husky and Hector-quadrotor start from B. Figure 13 shows that Clearpath Husky and AR Drone avoid two obstacles, while Turtlebot and Hector-quadrotor avoid only one obstacle before exiting the operating region. This is due to performance differences between TTC-OF and TTC-EO in estimating the distance to the obstacle. TTC-OF tends to have a higher error than TTC-EO and can lead to early obstacle avoidance, as indicated by the path travelled by Turtlebot and Hector-quadrotor. Since the obstacle avoidance is initiated early, the robots complete their turn manoeuvre and start moving straight early. This is the reason for different paths taken by the Turtlebot and Hector-quadrotor.

Experimental results

Figure 14 shows the layout of the environment for the real-time experiments. The operational environment is 4 m wide, and 14 m long. Turtlebot and AR Drone are used for real-time experiments. Turtlebot uses a Microsoft Kinect as vision sensor, while AR Drone utilizes its onboard camera. Both Microsoft Kinect and AR Drone camera generate images with 640×480 resolution. */odom* topic of Turtlebot (combination of wheel odometry and IMU) is utilized for Tutlebot position estimation, while localization of AR Drone is based on its */ardrone/odometry* topic (IMU). Figures 16 and 17 show the results for real-time experiments utilizing TTC-EO and TTC-OF obstacle avoidance algorithms, respectively. Both the robots start from the same starting location, and the starting location is changed collectively when obstacle avoidance algorithm is changed. Turtlebot and AR Drone travel with different velocities, with the velocity of AR Drone being higher. The experiments (obstacle avoidance using TTC-OF and TTC-EO) are repeated three times separately.

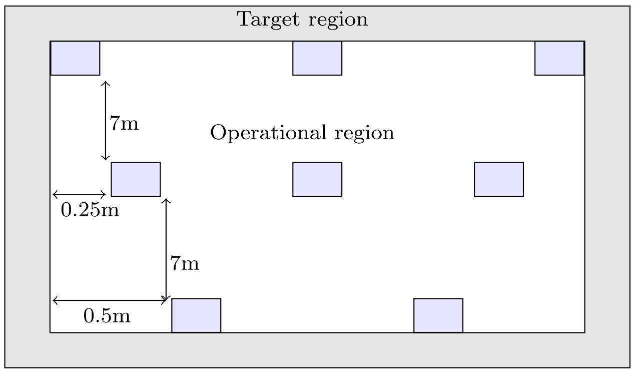
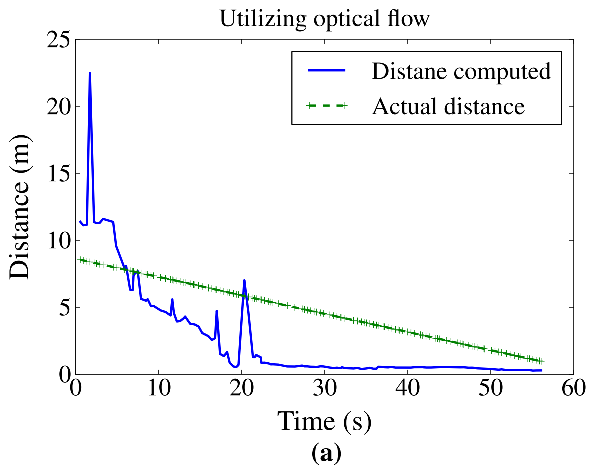


Fig. 14 Layout of the environment for real-time experiments (not drawn to scale). The *grey* region indicates the target region and the *white* region indicates the operational region. The mission is completed once the robot reaches the *grey* target region



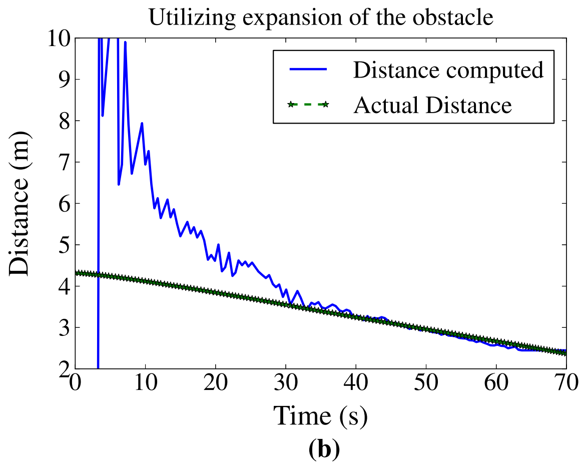


Fig. 15 Comparison between distance to the obstacle estimated utilizing TTC-OF and TTC-EO. **a** Estimated utilizing optical-flow-based time-to-contact algorithm (4), **b** estimated utilizing expansion-of-obstacle-based algorithm (7)

Figure 16 shows the best experimental results when TTC-EO is utilized for obstacle avoidance. Both Turtlebot and AR Drone start from the start location (0,0) and move forward (locations are expressed as Cartesian co-ordinates). Once the distance to the obstacle estimated is less than the critical distance, robots undergo turning manoeuvre (obstacle avoidance). It can be seen from Fig. 16 that both the robots avoid two obstacles before exiting the operation region. But AR Drone and Turtlebot travel in different directions while avoiding the first obstacle, as the location of the obstacle detected is different. Non-linear movement of AR Drone at the final stage of obstacle detection process results in different turning directions for the AR Drone. Figure 17 shows the best experimental results when TTC-OF is utilized for obstacle avoidance. Turtlebot and AR Drone start from the start location (14,1.75) and move towards the negative X-axis. AR Drone avoids two obstacles, one at (7,0) and the other at (−1.5,−2), while the turtlebot avoids one obstacle at (7,0), before exiting the operational region. Both the robots have the same turning direction, but the obstacle avoidance process is initiated earlier for the turtlebot. Figure 15 shows the distance to the obstace estimated utilizing TTC-OF (Fig. 15a) and TTC-EO (Fig. 15b). It can be seen that distance to the obstacle estimated utilizing TTC-EO follows the true distance after 4.5 m, while the distance to the obstacle estimated utilizing TTC-OF decreases closer to 3 m. Table 1 shows the mean error (ME), mean absolute-error (MAE) and mean-squared error (MSE) performances of TTC-EO and TTC-OF averaged over the three separate runs. To study the relation between the performance of TTC-EO and TTC-OF with respect to the distance to the obstacle, the error performances are analysed in ranges of 0.5 m. The error performance measures are calculated using the following:

$$\text{ME} = \frac{1}{n}\sum_{i=1}^{n}(\hat{d} - d_{\text{true}}) \tag{14}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|(\hat{d} - d_{\text{true}})| \tag{15}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{d} - d_{\text{true}})^2, \tag{16}$$

where n is the number of samples in the interval, $\hat{d}$ is the distance to the obstacle estimated and d_{true} is the true distance to the obstacle. It can be seen from the error measures that TTC-EO performs better compared to TTC-OF. This is evident from the simulation and experimental results (Figs. 12, 13, 16, 17). The error in distance to the obstacle estimated utilizing TTC-OF decreases as the robot moves closer to the obstacle as explained in [2]. It is noted that the ability of the Canny contours algorithm to detect the edges of the obstacle varies with respect to the lighting conditions (inherent problem with most of the vision based algorithms). The low and high thresholds of the Canny contour algorithm

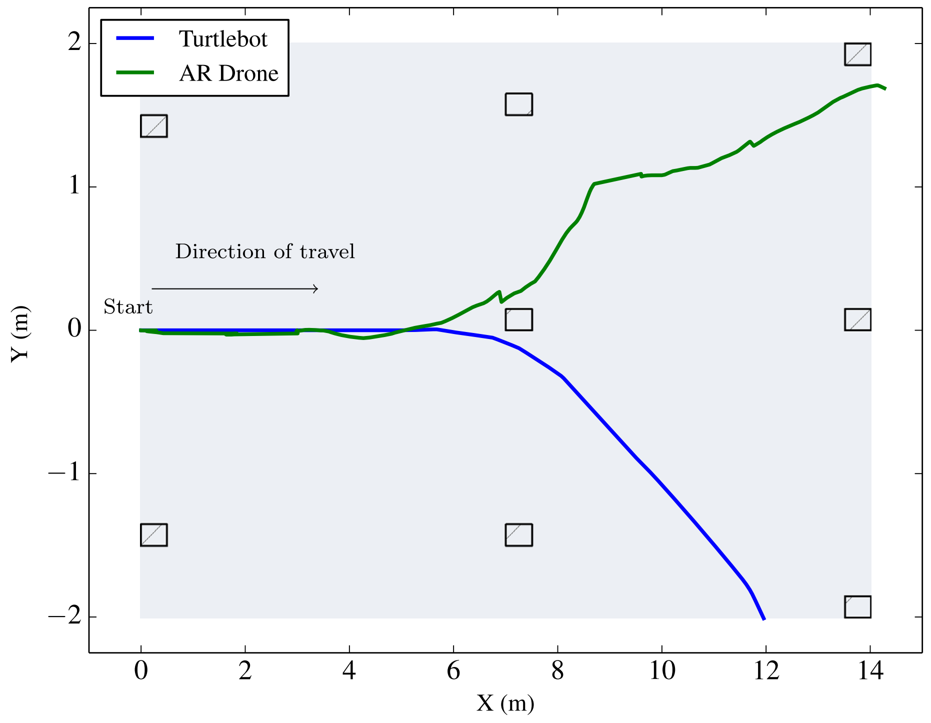


Fig. 16 *Top view* (bird's eye view) of the path taken by AR Drone and Turtlebot while completing the mission in real time. TTC-EO is utilized for obstacle avoidance. Each mission is carried out separately

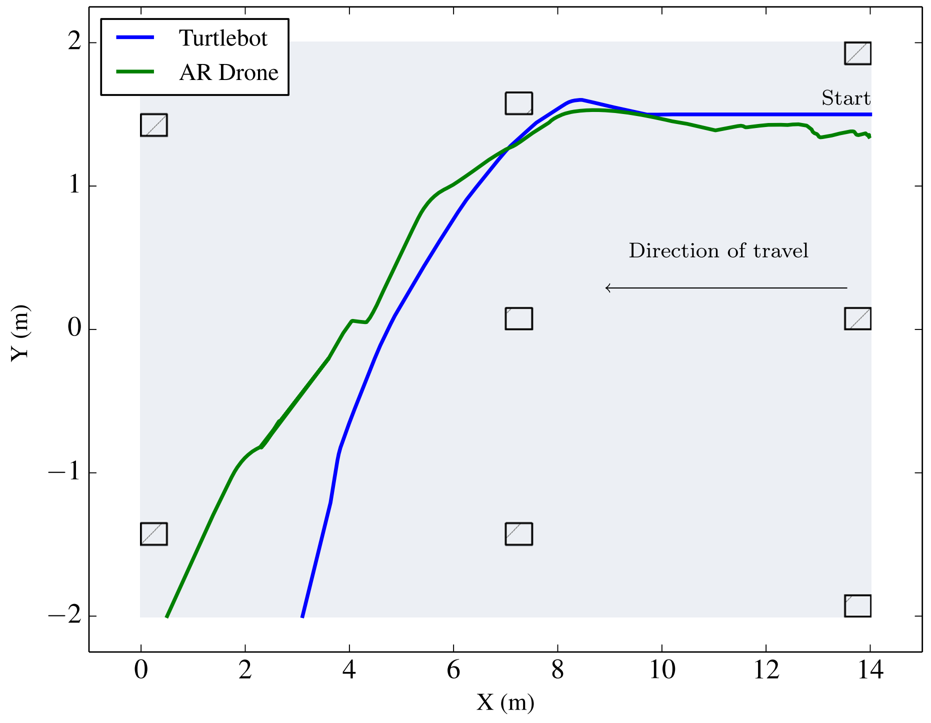


Fig. 17 *Top view* (bird's eye view) of the path taken by AR Drone and Turtlebot while completing the mission in real time. TTC-OF is utilized for obstacle avoidance. Each mission is carried out separately

needs to be changed depending on day/night lighting conditions.

In both simulation and real-time experiments, only the service for obstacle avoidance is modified. The rest of the services and the framework are not affected by this change. Similarly, the obstacle detection service may also be modified without affecting the rest of the framework. This demonstrates the modularity of the framework designed. Successful utilization of SOIFRA for collision avoidance on different platforms such as aerial vehicle (AR Drone and Hector-quadrotor) and ground vehicle (Turtlebot and Clearpath Husky) prove the interoperable nature of SOIFRA.

Conclusion and future works

This work presents an interoperable framework for autonomous robotic systems. The main focus of the SOIFRA framework proposed is to generalize platform-independent algorithms for unmanned aerial and ground robots. To demonstrate this, two obstacle avoidance algorithms (TTC-OF and TTC-EO) are utilized on aerial as well as ground robots. The behaviour-based nature of SOIFRA allows the agents to dynamically update their knowledge in real time, leading to effective collision avoidance. Service-oriented nature of SOIFRA helps to achieve this modularity, whereby a service can be replaced with other similar services without affecting other components of the framework. Though the control mechanisms for aerial robots and ground robots are completely different, the algorithms and mechanisms for obstacle detection and obstacle avoidance are generalized. SOIFRA utilizes this concept to achieve interoperability across diverse robotic platforms such as aerial and ground robots. Interoperability of SOIFRA is established by utilizing the framework for completing the same mission on four different robotic platforms (Turtlebot, Clearpath Husky, AR Drone and Hector-quadrotor) for simulations and two diverse robotic platforms (Turtlebot and AR Drone) for real-time experiments.

The scope for improvement in SOIFRA is multifold. The framework, at present, achieves collision avoidance for unmanned aerial and ground vehicles. There are many other standard requirements for autonomous robot operations such as path planning, path tracking, simultaneous localization and mapping. The framework is designed to make way for these additional requirements. Platform-independent path-planning and path-tracking algorithms can be easily incorporated as behaviours into the framework. Path generation would require a non-collaborative agent while path tracking would require a collaboration from obstacle avoidance and navigation agents. With the incorporation of pedestrian detection and tracking behaviours, SOIFRA can be used for surveillance, search and rescue operations, etc. This framework can also be improved to include autonomous underwater vehicles in addition to ground and aerial vehicles.

References

1. Alenya G, Nègre A, Crowley JL (2009) Time to contact for obstacle avoidance. European Conference on Mobile Robotics
2. Arokiasami WA, Chen TK, Srinivasan D, Vadakkepat P (2015) Impact of the length of optical flow vectors in estimating time-to-contact an obstacle. In: Proceedings of the 18th Asia Pacific symposium on intelligent and evolutionary systems, vol 2. Springer, Berlin, pp 201–213
3. Badano BMI (2008) A multi-agent architecture with distributed coordination for an autonomous robot. PhD thesis
4. Berenz V, Tanaka F, Suzuki K, Herink M (2011) Tdm: a software framework for elegant and rapid development of autonomous behaviors for humanoid robots. In: 11th IEEE-RAS international conference on humanoid robots (Humanoids), 2011, IEEE, pp 179–186
5. Browning NA (2012) A neural circuit for robust time-to-contact estimation based on primate mst. Neural Comput 24(11):2946–2963
6. Chen JY (2010) Uav-guided navigation for ground robot teleoperation in a military reconnaissance environment. Ergonomics 53(8):940–950
7. Corke P, Hrabar S, Peterson R, Rus D, Saripalli S, Sukhatme G (2004) Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In: Proceedings of the ICRA'04, 2004 IEEE international conference on robotics and automation, IEEE, vol 4, pp 3602–3608
8. de Croon GC (2016) Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. Bioinspiration Biomim 11(1):016004
9. Doherty P, Kvarnstrom J, Wzorek M, Rudol P, Heintz F, Conte G (2015) HDRC3: a distributed hybrid deliberative/reactive architecture for unmanned aircraft systems. In: Valavanis KP, Vachtsevanos GJ (eds) Handbook of unmanned aerial vehicles. Springer, Netherlands, pp 849–952
10. Elfes A, Dolan JM, Podnar G, Mau S, Bergerman M (2006) Safe and efficient robotic space exploration with tele-supervised autonomous robots. In: AAAI spring symposium: to boldly go where no human-robot team has gone before, pp 104–113
11. Fox D, Burgard W, Thrun S (1997) The dynamic window approach to collision avoidance. IEEE Robot Autom Mag 4(1):23–33
12. Green WE, Oh PY (2008) Optic-flow-based collision avoidance. Robot Autom Mag IEEE 15(1):96–103
13. Hennes D, Claes D, Meeussen W, Tuyls K (2012) Multi-robot collision avoidance with localization uncertainty. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems, Vol 1, pp 147–154
14. Hennes D, Claes D, Meeussen W, Tuyls K (2012) Multi-robot collision avoidance with localization uncertainty. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems, vol 1, International foundation for autonomous agents and multiagent systems, pp 147–154
15. Hsu HH, Liu A (2007) A flexible architecture for navigation control of a mobile robot. IEEE Trans Syst Man Cybern Part A Syst Hum 37(3):310–318
16. Insaurralde CC, Petillot YR (2015) Capability-oriented robot architecture for maritime autonomy. Robot Auton Syst 67:87–104
17. Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. Int J Robot Res 5(1):90–98

18. Ku CH, Tsai WH (1999) Obstacle avoidance for autonomous land vehicle navigation in indoor environments by quadratic classifier. IEEE Trans Syst Man Cybern Part B Cybern 29(3):416–426
19. Kunz C, Murphy C, Camilli R, Singh H, Bailey J, Eustice R, Jakuba M, Nakamura KI, Roman C, Sato T et al. (2008) Deep sea underwater robotic exploration in the ice-covered arctic ocean with auvs. In: IEEE/RSJ international conference on intelligent robots and systems, 2008, IROS 2008, IEEE, pp 3654–3660
20. Lesser VR (1999) Cooperative multiagent systems: a personal view of the state of the art. IEEE Trans Knowl Data Eng 11(1):133–142
21. Li K, Zhang Q, Kwong S, Li M, Wang R (2014) Stable matching-based selection in evolutionary multiobjective optimization. IEEE Trans Evol Comput 18(6):909–923
22. Li X, Mabu S, Hirasawa K (2014) A novel graph-based estimation of the distribution algorithm and its extension using reinforcement learning. IEEE Trans Evol Comput 18(1):98–113
23. Liu J, Wu J (2001) Multiagent robotic systems. CRC Press, Boca Raton
24. Lucas BD, Kanade T et al (1981) An iterative image registration technique with an application to stereo vision. IJCAI 81:674–679
25. Maza I, Caballero F, Capitán J, Martínez-de Dios J, Ollero A (2011) Experimental results in multi-uav coordination for disaster management and civil security applications. J Intell Robot Syst 61(1–4):563–585
26. Muratet L, Doncieux S, Briere Y, Meyer JA (2005) A contribution to vision-based autonomous helicopter flight in urban environments. Robot Auton Syst 50(4):195–209
27. Nesnas IA, Wright A, Bajracharya M, Simmons R, Estlin T (2003) Claraty and challenges of developing interoperable robotic software. In: Proceedings of the 2003 IEEE/RSJ international conference on intelligent robots and systems, (IROS 2003), vol 3, pp 2428–2435
28. Nwana HS (1996) Software agents: an overview. Knowl Eng Rev 11(03):205–244
29. O'Donovan P (2005) Optical flow: techniques and applications. The University of Saskatchewan, TR 502425
30. Rezaei M, Saghafi F (2011) Optical flow-based obstacle avoidance of a fixed-wing mav. Aircr Eng Aerosp Technol 83(2):85–93
31. Rockel S, Klimentjew D, Zhang J (2012) A multi-robot platform for mobile robotsa novel evaluation and development approach with multi-agent technology. In: 2012 IEEE Conference on Multisensor fusion and integration for intelligent systems (MFI), pp 470–477
32. Sabo C, Cope A, Gurny K, Vasilaki E, Marshall JA (2016) Bio-inspired visual navigation for a quadcopter using optic flow. In: AIAA Infotech@ Aerospace, p 0404
33. Sišlák D, Volf P, Komenda A, Samek J, Pechouček M (2007) Agent-based multi-layer collision avoidance to unmanned aerial vehicles. In: International Conference on integration of knowledge intensive multi-agent systems, 2007. KIMAS 2007, IEEE, pp 365–370
34. Tammero LF, Dickinson MH (2002) The influence of visual landscape on the free flight behavior of the fruit fly drosophila melanogaster. J Exp Biol 205(3):327–343
35. Tamminga A, Hugenholtz C, Eaton B, Lapointe M (2015) Hyper-spatial remote sensing of channel reach morphology and hydraulic fish habitat using an unmanned aerial vehicle (uav): a first assessment in the context of river research and management. River Res Appl 31(3):379–391
36. Tigli JY, Thomas M (1994) Use of multi agent systems for mobile robotics control. In: 1994 IEEE international conference on systems, man, and cybernetics, 1994. Humans, information and technology, vol 1, pp 588–592
37. Tistarelli M, Grosso E, Sandini G (1991) Dynamic stereo in visual navigation. In: Proceedings of the CVPR'91., IEEE computer society conference on computer vision and pattern recognition, 1991, pp 186–193
38. Ulrich I, Borenstein J (2000) VFH*: Local obstacle avoidance with look-ahead verification. In: ICRA, pp 2505–2511
39. Vadakkepat P, Miin OC, Peng X, Lee TH (2004) Fuzzy behavior-based control of mobile robots. IEEE Trans Fuzzy Syst 12(4):559–565
40. Vallejo D, Remagnino P, Monekosso DN, Jiménez L, González C (2009) A multi-agent architecture for multi-robot surveillance. In: Nguyen NT, Kowalczyk R, Chen S-M (eds) Computational collective intelligence. Semantic web, social networks and multiagent systems. Springer, Berlin, Heidelberg, pp 266–278
41. Waibel M, Keller L, Floreano D (2009) Genetic team composition and level of selection in the evolution of cooperation. IEEE Trans Evol Comput 13(3):648–660
42. Willms A, Yang SX (2006) An efficient dynamic system for real-time robot-path planning. IEEE Trans Syst Man Cyber Part B Cyber 36(4):755–766
43. Yeh CH, Yang CY (2015) Social networks and asset price dynamics. IEEE Trans Evol Comput 19(3):387–399
44. Zingg S, Scaramuzza D, Weiss S, Siegwart R (2010) Mav navigation through indoor corridors using optical flow. In: 2010 IEEE international conference on robotics and automation (ICRA), pp 3361–3368