AL-Kafaf *et al. Complex Adapt Syst Model* (2018) 6:8
https://doi.org/10.1186/s40294-018-0059-1

Complex Adaptive Systems Modeling

RESEARCH Open Access

A three-phase decision making approach for self-adaptive systems using web services

Dhrgam AL-Kafaf*†, Dae-Kyoo Kim† and Lunjin Lu†

*Correspondence:
dalkafaf@oakland.edu
†Dhrgam A. L. Kafaf, Dae-Kyoo Kim and Lunjin Lu contributed equally to this work
Computer Science Department, Oakland University, 115 Library Drive, Rochester, MI 48309-4479, USA

Abstract

A self-adaptive system adapts itself to changes in a dynamic environment. The core in self-adaptive systems is making an adaptation decision. The current practice focuses on a single layer of decision making using either a local knowledge base or a shared knowledge base shared by multiple units through a network. While the use of a local knowledge base is efficient, it suffers from its limited maturity. A shared knowledge base can address the maturity problem, but it is inefficient in adaptation due to communication overheads. In this work, we present a three-phase decision making approach for self-adaptive systems to improve precision while being competitive for efficiency. The approach consists of three phases for making a decision. The first phase uses the local knowledge base of the self-adaptive unit to identify an object. If the object cannot be identified locally, the unit sends a request to shared knowledge bases through web services in the second and third phases. The approach makes use of B-kNN for object identification and web services for accessing shared knowledge bases. We conducted quantitative validation in terms of accuracy, precision, recall, and F-measure using a set of scenarios. The results show that 99.58% of accuracy, 94.01% of recall, 94.04% of precision, and 94.01% of F-measure can be achieved. We also conducted comparative analysis by comparing the presented approach with the traditional approach and the cloud-based approach. The results show that the presented approach improves 45% in object identification with an increase of 0.66 s over the traditional approach and the same performance in object identification with a decrease of 0.95 s over the cloud-based approach.

Keywords: Adaptive systems, Unmanned ground vehicle, Machine learning, B-kNN

Introduction

There has been a growing interest in self-adaptive software systems for their various use in different domains such as self-driving vehicles. The core of self-adaptive systems is the decision-making process which is based on a knowledge base. The knowledge base evolves as the system experiences more adaptations. However, the current practice mainly relies on a single-phase decision making using either the local knowledge base in the system (e.g., Garlan et al. 2003; Wang and Silva 2008; Dorigo and Schnepf 1993) or an external knowledge base shared by multiple systems (e.g., Garlan et al. 2003; Cheng et al. 2005; Kramer and Magee 2007; Bonaccorsi et al. 2015; Kehoe et al. 2013). A local knowledge base system can make a fast adaptation decision, but precision is low due to the limited evolution by single learning. A shared knowledge-base system shares

a knowledge-base with other systems through a network. The knowledge base can be highly mature due to the collaborative contribution to the evolution of the knowledge base, which enables to make a precise decision. However, a major concern is possibly delayed or failed adaptations due to network overheads or failure which is critical in a real-time environment.

In this paper, we present a three-phase decision making approach for self-adaptive vehicle systems to improve the precision of decisions with competitive efficiency. The approach makes use of B-kNN (Kafaf et al. 2017), a variation of k-Nearest Neighbors (kNN) (Cover and Hart 1967) for improved efficiency in object identification. In the approach, a decision is made in three phases supported by web services. In the first phase, a self-adaptive unit (SAU) makes the initial decision for an encountered object using its local knowledge base. If the object cannot be identified, the SAU sends a request to a context-specific knowledge base, which is specific to an environment (e.g., indoor, outdoor), in the second phase via a web service. The context-specific knowledge base is shared and evolves by multiple SAUs within the same environment. If the object still cannot be identified, the system further requests to the global knowledge base in the third phase. The global knowledge base is shared and contributed by all vehicles across different environments.

We implemented the approach using Robot Operating System (ROS) (Quigley et al. 2009) which is a software framework providing operating system-like functionality for developing robotic software. For web services, we use Representational State Transfer (REST) (Richardson and Ruby 2008) which is an architectural style for networked hypermedia applications. We evaluated the implementation using Gazebo (Koenig and Howard 2004) which provides a 3D simulation environment for robotic systems. The evaluation is carried out for four different scenarios with the results of each scenario analyzed in terms of precision and response time. The evaluation shows 96% precision in object identification with viable overheads introduced by the web service and 45% improvement in precision over the traditional approach which relies on the local knowledge base only. This research is an advance of our previous work (Kafaf and Kim 2017) by improving precision and efficiency using B-kNN. The improvement of efficiency is also contributed by REST web services instead of SOAP web services used in the previous work.

The remainder of the paper is organized as follows. "Related work" section discusses works from the literature on self-adaptive systems. "Three-phase decision making process" section describes this work approach. "Validation" section evaluates the approach using a set of scenarios in terms of quantitative and comparative analysis. "Conclusion" section concludes with discussions on the future work.

Related work

The work by Magee and Kramer (1996) on dynamic modeling of software architecture has inspired many subsequent works (e.g., see Garlan et al. 2003; Cheng et al. 2005; Kramer and Magee 2007; Bonaccorsi et al. 2015; Kehoe et al. 2013) on feedback loops which are essential for the evolution of self-adaptive systems. However, the feedback mechanism in these works is often kept hidden or abstract. IBM Autonomic Computing Architecture (Kephart and Chess 2003) introduced MAPE, an open architecture for

self-adaptive systems that consists of monitoring, analyzing, planning, and executing components.

Garlan et al. (2003) presents a set of conditions for monitoring environmental properties (e.g., network latency) for a system to dynamically adapt to environmental changes. A condition is defined in terms of operations and repair strategies specific to an architectural style. The conditions are specified at the architectural level for generality and simplicity. However, their approach is limited in learning and adapting to a situation that is not considered in the conditions.

Wang and Silva (2008) presents a machine-learning approach for developing a multi-robot system where a group of intelligent robots work cooperatively to transport an object to a goal location in a dynamic environment. Their approach integrates reinforcement learning (RL) with genetic algorithms (GAs) in order to increase the precision of system behaviors. However, RL and GAs are slow in adapting to a dynamic environment, which is an inherent limitation on their approach. Furthermore, the overhead introduced by the integration offsets the precision improvement resulting from integration. They also present a modified Q-learning algorithm for making a decision when conflicts arise in resource use or behaviors.

Dorigo and Schnepf (1993) present an approach for developing behavior-based robots using multiple classifier systems running in parallel in a hierarchical structure. Each classifier system learns simple behaviors through interactions with the environment. The hierarchical organization distinguishes two learning activities, one for learning behavioral sequences and another for learning coordination sequences. Classifier systems at the lowest level in the hierarchy learn behavioral sequences which are real actions activated by sensory input from the environment. Only the classifier systems at the lowest level have direct access to the environment via the sensors and actuators of the robot. On the other hand, classifier systems at a higher level learn to coordinate the activities of classifier systems at lower levels. Their work is modular, allowing more classifier components to be added to learn more behaviors. However, that is at the expense of increased resource consumption.

More recently, several researchers (e.g., Hickman et al. 2014; Liu et al. 2014; Tian et al. 2015; Hu et al. 2012) propose to use cloud computing as a base platform for the knowledge base of robotic systems. The use of the cloud enables a robotic system to be more scalable and efficient, while allowing it to be lighter and smaller in size. This also improves the performance of identifying objects by the increased maturity of the knowledge base shared and contributed by multiple robotic units through the cloud.

Kuffner et al. (2014) introduced the use of a cloud-based knowledge base for collecting and analyzing information about an encountered object and determining robot behaviors in response to the object. The knowledge base receives a feedback from the robot on every interaction with the object. The knowledge base may benefit other robots facing the same experience. Unlike our work using a two-phase decision process, their approach solely relies on the cloud for determining robot behaviors.

Liu et al. (2014) proposed a cloud-based architecture for distributed robotic information fusion systems to offload computation to the cloud. The architecture is based on ROS, consisting of one master controlling multiple robots simultaneously in the network where the master is loaded into a virtual machine. In their approach, a failure of the ROS

master causes the halt of the entire system, which raises a reliability concern. Unlike their approach, we allow a self-adaptive unit to have its own local knowledge base, which enables the system to continuously run even in the case where the web service is unavailable for any reason. Similar to Kuffner et al.'s work (Hickman et al. 2014), they also offload image processing into the cloud. However, relying solely on the could for image processing introduces significant communication overheads.

Hu et al. (2012) proposed a cloud-based architecture for robotic systems to enable the extension of the computation and information sharing of robots on a network. The architecture leverages the combination of an *ad-hoc* cloud formed by machine-to-machine (M2M) communications among participating robots and an infrastructure cloud enabled by machine-to-cloud (M2C) communications. However, it is not clear how the architecture can be validated as there are no details on implementation and experiments.

Chen et al. (2010) introduced the concept of "Robot as a Service (RaaS)" which is an all-in-one design for the service provider, service broker, and service client to support robot services performed remotely in different places, which reduces computation loads by distributing them to multiple robots in the network. However, similar to Liu et al.' work, their approach also exhibits overhead issues with respect to the response time of units.

Tian et al. (2015) proposed a cloud computing platform based on an intelligent space where a user interacts with computers and robots to use their services. In their work, the cloud is used to extend the knowledge of robots when a requested service is not available locally. Our work is similar to their work in that both local and external knowledge bases are used. However, it is not clear in their work how a robot should behave when an object is encountered. Also, there is little validation conducted in their work.

Bonaccorsi et al. (2015) presented a cloud-based design for mobile robots that help seniors for reminding medication, monitoring remote indoor, and locating people indoor in case of falling. The design is based on two modules—RaaS and SaaS. RaaS is responsible for operating the robot and managing wireless sensor networks, while SaaS is responsible for locating people indoor and managing the calendar and database. The operations of finding people and reminding medication are outsourced to the cloud, which offloads the computation from the robot. However, the reliability of these operations is limited and the robot may not perform any operation in case of communication difficulty, which is a major concern in relying solely on the cloud.

Kehoe et al. (2013) presented an architecture for Cloud-based robots using Google Goggles which is an object recognition engine. The architecture is two-fold—offline and online phases. The offline phase is for training when robots or humans collect objects images and pre-process them and upload the data to a custom version of Google Goggles. In the online phase, a robot solely depends on Google Goggle online for object recognition. Their work has a similar problem as the work by Bonaccorsi et al. that a communication failure can be critical.

In our previous work (Kafaf and Kim 2017), we presented a two-phase approach based on the traditional kNN algorithm for making a self-adaptive decision using SOAP for web services. The work in this paper improves the previous work by using a three-phase approach based on B-kNN (with statistical threshold detection) using REST for web

services to improve precision and efficiency. The overhead introduced by the additional phase is outweighed by the efficiency gained by B-kNN and REST as discussed in "Validation" section.

Three-phase decision making process

Self-adaptive decision making has gained significant interests in various types of applications in different domains such as personal/professional service robots in the service robot domain, smart solder helmets in the military domain, and autonomous vehicles in the automotive domain. A major concern in these applications is to identify a wide range of objects with high precision and efficiency.

In this section, we describe the three-phase decision making process for self-adaptive systems. The approach aims to improve the object identification in a self-adaptive system with competitive efficiency. The application of the technique includes personal/professional service robots in the service robot domain, smart solder helmets in the military domain, and autonomous vehicles in the automotive domain. The process consists of three phases. In the first phase, the SAU makes the initial decision to identify an encountered object using its local knowledge base (LKB). If the object cannot be identified, the SAU sends a request to a context-specific knowledge (CKB) base through a web service in the second phase. The CKB is shared and evolves by multiple SAUs in the same environment (e.g., indoor). As the CKB is contributed by multiple SAUs, it is more mature than local knowledge bases. There are multiple CKBs for different contexts and the SAU determines which one to use based on its assessment of the given image. If the object still cannot be identified by the CKB, the CKB delegates the request to the global knowledge base (GKB) which is an aggregate of CKBs. The GKB assesses the received image to determine the context. If the context is different from the one of which the requesting CKB is concerned, the GKB forwards the request to the CKB of the determined context. If the second CKB cannot identify the object either, the CKB send the request back to the GKB and the GKB searches for the object within its dataset. If the object cannot be identified by the GKB, the GBK creates a new class for the object. Figure 1 depicts the three-phase decision making process where CKBs and the GKB reside in the web.

In the remaining section, we describe B-kNN for identifying objects and the architectural components in the three-phase decision making process.

B-kNN

In this work, we use B-kNN (Kafaf et al. 2017) which is a variation of kNN (Cover and Hart 1967) for improved efficiency in object identification. While the traditional kNN is exhaustive in searching, B-kNN is selective, which reduces searching time. Exhaustive search is more precise, but less efficient, which can be critical when datasets are enormous. B-kNN is designed to be efficient with competitive precision. B-kNN preprocesses the training dataset to reduce computational time by selecting a set of representative points from the training dataset. The set is defined based on the minimum and maximum points (MMP) for each class in the training dataset in order to form a boundary of the class. The MMP are determined by the minimum and maximum value of each dimension (attribute) of the class. The set of points that fall on the boundary of a class referred to as boundary set (BS) represents the class and

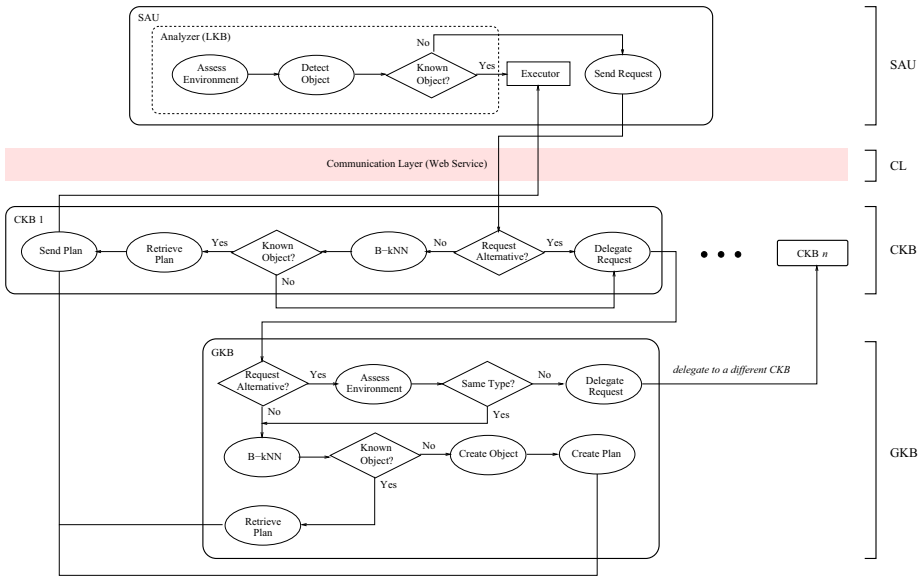

Fig. 1 Three-phase decision making

the set of the BSs of the training datasets represents the training dataset. Overlapping boundaries may result in less accurate classification when the test element exists in the overlap. This is addressed by adjusting the MMP (e.g., decreasing the minimum point, increasing the maximum point). Algorithm 1 shows the algorithm of B-kNN. In the algorithm, the test element is examined to the MMP of classes. If it is found in any MMP, the element has been identified. If not, the element is examined to the BS of classes using kNN to determine the nearest class which becomes the class of the element. We adopt Grubbs' test (Grubbs 1969) to set thresholds to determine the class membership of the test element. A threshold is set for each class to reflect the relative distribution of the elements in the class. If the distance of the test element to a class is smaller than the threshold of the class, the test element is determined to belong to the class.

Algorithm 1 B-kNN Algorithm

```
1: procedure PREPROCESSING(TD:in Training Dataset)
2:     Separate Classes (TD)
3:     MMP ← Define MMP of CLass (TD)
4:     BS←Define BS of Class (TD)
5: end procedure
6: procedure B-KNN(TD:in Training Dataset,TE:in Testing Element, EC:out Element class)
7:     MMP ← Preprocessing (TD)
8:     BS← Preprocessing (TD)
9:     for Each Class do
10:        if TE ∈ MMP then
11:            EC ← Class Type
12:        end if
13:    end for
14:    if TE ∉ MMP then
15:        EC ← kNN(TE, BS)
16:    end if
17: end procedure
```

Architectural components

The three-phase decision making process involves two major components—SAU and web service where SAU is concerned with making the initial decision using its LKB and the web service is concerned with making the second and third decisions using a CBK and the GBK, respectively.

Self-adaptive unit (SAU)

An SAU represents a single self-adaptive unit. It moves around in the given environment and adapts itself to an encountered challenge (e.g., obeying traffic signs). Figure 2 shows the model of a SAU based on the IBM model (Kephart and Chess 2003). It consists of a monitor, an analyzer, a local planner, and an executor. The monitor receives as input images from the camera mounted on the SAU and the speed of the SAU from the actuators of the wheels. The inputs are sent to the analyzer to determine instructions to be executed. The instructions are converted to an action by the local planner and executed by the executor.

This research uses the Kinematic model (Haug 1989) to control the state change of a SAU and calculate the direction and speed of the SAU. The dynamics model (Bleicher et al. 1999) is another model that can be used for the same purpose, but requires more parameters (e.g., material properties, the inertia of wheels) which are not concerned in this work. Figure 3 shows the Kinematics model for a two-wheeled differential-drive vehicle that has an independent actuator on each wheel. In the model, the circle represents the robot, the two boxes on the left and right represent the wheels, and the arrows denote the rotation direction. The letter W specifies the width of the robot, V specifies the velocity of the vehicle, V_{tl} and V_{tr} denote the velocity of the left and right wheel respectively, and ψ specifies the rotation degree.

Monitor The monitor receives images from the camera on the SAU and the RPM of wheels from the actuators on the wheels. Images are used to identify the surroundings of the SAU and RPMs are used to determine the current speed of the SAU. The orientation of the SAU is determined by the Kinematic model. Given the information, the monitor calculates the distance of the SAU from an encountered object. The collected and computed information is stored and sent to the analyzer to determine an appropriate reaction to the object. Algorithm 2 describes the monitoring operation. The operation receives

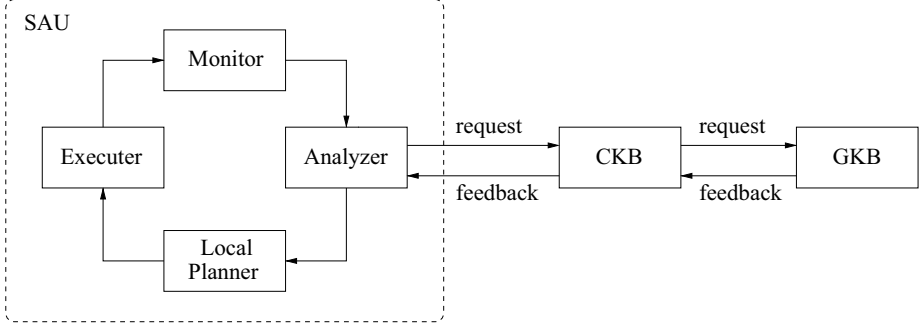

Fig. 2 Self-adaptive feedback loop

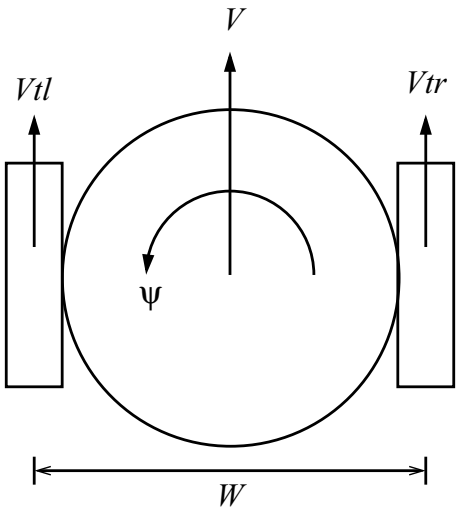

Fig. 3 Differential-drive vehicle kinematics (Haug 1989)

camera images, vehicle speed, and LiDAR streams and calculates the distance between the SAU to the encountered object.

Algorithm 2 Monitoring Operation

```
1: procedure MONITORING
2:     while True do
3:         Image ← Retrieve Image ()
4:         VS ← Retrieve Vehicle Speed ()
5:         LiDAR ← Retrieve LiDAR Stream ()
6:         Calculate Distance (LiDAR)
7:         Analyzer (Image, VS)
8:     end while
9: end procedure
```

Analyzer Based on the data received from the monitor, the analyzer identifies the environmental objects surrounding the SAU using the LiDAR on the SAU to avoid collision. Algorithm 3 describes the Analyzer operation. It first identifies the type of the environment to narrow down object types that might be encountered. We consider three types of environments—indoor, urban, and rural. For example, if the environment is identified as outdoor, typical outdoor objects such as traffic signs are expected and considered first in the identification process. This helps improve efficiency by reducing the problem space, which is specially helpful SAUs whose computing resources are limited.

Algorithm 3 Analyzing Operation

```
1: procedure ANALYZING(Image:in Camera Image, VS: in Vehicle Speed)
2:     while True do
3:         SEnv ← Detect Environments (Image)
4:         if SEnv Object ∈ Detect Object (Image) then
5:             Local Planner (Object, VS)
6:         else
7:             Send to SKB (Object)
8:         end if
9:     end while
10: end procedure
```

After determining the environment type, the analyzer identifies the detected object using the LKB. If the object is successfully identified, it is sent to the local planner to determine an adaptive behavior based on the action plans that are locally available in the SAU. If the object cannot be identified by the LKB, the SAU sends an identification request to the CKB that is specific to the environment type through a web service. If a plan is not received until the SAU reaches 0.25 m from the object, the adaptation fails and the SAU makes a 180° turn to avoid the collision. If the received plan is found to be unsuitable and there is sufficient time to make another request before reaching the turning point, the SAU sends a feedback to the CKB and requests an alternative plan. The CKB delegates the feedback and request to the GKB. The GKB then selects an alternative plan and sends it back to the CKB. The CKB updates its own dataset with the received plan and delegates the plan to the requesting SAU. If the alternative plan is not suitable either with no sufficient time to make another request, the adaption fails and the SAU makes a 180° turn.

Local planner The local planner controls the speed of the SAU to support the object identification process in the analyzer. Algorithm 4 describe the planning operation. While the analyzer identifies an object, the local planner slows down as it approaches the object, allowing time for the analyzer to communicate with the CKB. This also helps to prevent a possible collision caused by a radical change of speed. The adaptation plans vary depending on the type of the encountered object. For example, the SAU makes a complete stop for three seconds if the object is a stop sign, or the SAU avoid collision with pedestrians. This research also defines three types of distances—near, fuzzy, and far to mandate a different reaction in speed depending on the distance of the SAU to the encountered object. In any of these cases, the SAU tries to make an adaptation decision until reaching the minimum distance to the object to avoid a collision. Objects that are small enough to pass under the vehicle's wheels are not considered in this work.

Algorithm 4 Local Planner Operation

```
1: procedure LOCAL PLANNER(ObInf:in Object Information, VS: in Vehicle Speed)
2:     while True do
3:         if ObInf ← true then
4:             Plan ← Retrieve Plane (ObInf)
5:         end if
6:         Distance ← Retrieve Distance()
7:         if Distance ∈ Near then
8:             Abort Mission ()
9:         else if Distance ∈ Fuzzy then
10:            Reduce Speed()
11:            Execution (Plan, VS)
12:        else if Distance ∈ Far then
13:            Execution (Plan, VS)
14:        end if
15:    end while
16: end procedure
```

Executor The executor performs the adaptive operation (e.g., turn left, turn right) that is determined by the local planner. It converts the operation into an actual action to be carried on the SAU's wheels through the Kinematic model. It first determines if the operation requires the SAU to turn. If so, it calculates the degree of the turn based on the current direction of the SAU and the distance to the object. The individual

wheels of the SAU run on differential drive which allows them to have different speeds to make a turn. The turning speed of a wheel is computed based on the angle of the turn and sent to the actuator of the wheel. The SAU is designed to make a 180° turn when reaching 0.25 m to the object in order to avoid a collision. Algorithm 5 describes the execution operation.

Algorithm 5 Execution Operation

```
1: procedure EXECUTION(Plane:in Execution Plan, VS: in Vehicle Speed)
2:     while True do
3:         SAU Heading ← Retrieve Heading ()
4:         WS ← Calculate Wheel Spead (Plan, SAU Heading, VS)
5:         Send WS to Wheels()
6:     end while
7: end procedure
```

Web services

For those objects that cannot be identified by the LKB of the SAU, the SAU sends a request through a web service to the CKB that is specialized for the identified environment type. If the CKB is also unable to identify the object, it further delegates the request to the GKB. Note that CKBs and the GKB reside in the same server. We use a web service for the communication between the LKB and the CKB. web service, which is a standard protocol, is chosen for its easiness of implementation and low cost of communication.

Context-specific knowledge base

When a request is received, CKB determines whether the object has been previously identified by other SAUs sharing and contributing to the CKB. CKBs are also based on B-kNN and the identification process is the same as LKBs. If the object is already known to the CKB, it retrieves the existing adaptation plan associated with the object from the data storage and sends it to the SAU. If the object cannot be identified, the CKB delegates the request to the GKB for more comprehensive search. Algorithm 6 describes the CKB Operation. Note that CKBs evolve by updating their dataset with responses received from the GKB for unknown objects.

Algorithm 6 Context Specific Knowledge Base (CKB) Operation

```
1: procedure SKB(Image :in Object Image, Plan: out Execution Plan)
2:     if Feedback ← True then
3:         Send to GKB
4:     elseDetect Object
5:         if Object ∈ Detect Object (Image) then
6:             Plan ← Retrieve Plane (Object)
7:             Send Plan to SAU
8:         else
9:             Send to GKB
10:        end if
11:    end if
12: end procedure
```

Global knowledge base

When receiving the request, the GKB reassesses the environment type of the object with a more mature dataset. If the GKB determines that the object belongs to a different environment from that of the requesting CKB, the GKB delegates the request to the

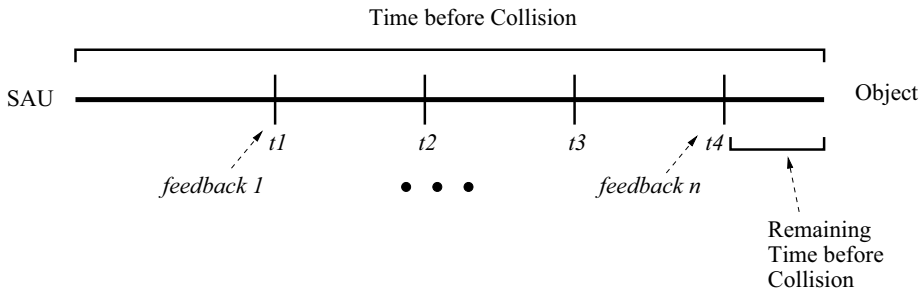

Fig. 4 Feedback

corresponding CKB. If the CKB cannot identify the object either, it returns the request to the GKB. The GKB then checks whether the object has been previously identified in its dataset. If not, it determines the class membership for the object using B-kNN and sends the adaptation plan associated with the class to the requesting CKB which forwards the response to the requesting SAU. If the GKB determines the environment type to be the same as the type of the requesting CKB and the object is known to the GKB, the GKB sends the class membership and its associated adaptation plan to the requesting CKB.

If the object is unknown to the GBK, the GKB defines a new type for the object using the k-means clustering based on their similarities. The new type is defined in terms of name and adaptation plan. The name should be unique from existing types to avoid name conflicts. An adaptation plan is determined for the new type by k-means and B-kNN. k-means first determines the k nearest type for the object. The k-nearest type is inserted into B-kNN to retrieve its associated adaptation plan. The plan is then set as the plan for the new type. If the plan fails on the SAU, the SAU sends a feedback to the CKB which delegates the request to the GKB for an alternative plan. This process continues until the SAU reaches the turning point to avoid a collision. If no plan succeeds within the viable time, the SAU sends a failure feedback to the CKB which delegates the feedback to the GKB. The CKB and GKB then de-associate the failed plan from the object type in their dataset. Algorithm 7 describes the GKB operation.

The SAU may continue to request an alternative plan while there is enough time left before colliding with the object. Figure 4 illustrates the time line for feedbacks. If received adaptations continue to fail and there is insufficient time remaining to receive another plan, the SAU stops moving and makes a 180° turn.

Algorithm 7 General Knowledge Base (GKB) Operation

```
1: procedure GKB(Image :in Object Image, Plan: out Execution Plan)
2:     if Env ∈ Reassess Environment () then
3:         if Object ∈ Detect Object (Image) then
4:             Plan ← Retrieve Plane (Object)
5:             Send Plan to SAU
6:         else
7:             Object ← Define New Object (Image)
8:             Plan ← Define New Plan (Object)
9:             Send to SAU
10:        end if
11:    else
12:        Send to SKB
13:    end if
14: end procedure
```

Validation

We validated the approach using Gazebo (Koenig and Howard 2004) which is a robot simulation framework for design and testing of robotic systems in various settings of scenarios. We use Gazebo as an outdoor simulator which we found is suitable for this work. However, by the nature of simulators, it comes with limitations in physical aspects such as simulating extreme weather conditions or sun light reflection on road lane marking (Koenig and Howard 2004) which are not considered in this work.

We implemented SAUs on Robot Operating System (ROS) (Quigley et al. 2009) which is an open source framework for development of robot software. ROS was chosen for its portability in various operating systems and ample documentations. SAUs are populated in a simulation environment created by Gazebo. Each SAU is equipped with a camera and LiDAR. A SAU uses a VGA camera that has 640 $\times$ 480 resolution with the shutter speed of 20 frames/s as the main input sensor for monitoring the surroundings. The resolution is high enough to recognize an object boundary, while requiring less computation resources for image processing. We also used LiDARs for detecting surrounding objects and measuring the distance of the SAU to an object. LiDARs allow for more accurate measuring of object distance than cameras, while consuming less computing resources. LiDARs are supported by the *move_base* ROS package in Gazebo which provides a route plan to reach a given destination. Gazebo was run on a 64-bit operating system with Intel Core i7 CPU (2.4 GHz) and 8 GB RAM. We also used Open Source Computer Vision Library (OpenCV) (Bradski and Kaehler 2008) for image processing. For web services, we used REST for efficient communication.

Figure 5 shows a screen capture of the validation environment in Gazebo. In the figure, the left window is used to configure the environment through the *World* tab which lists the objects (e.g., tables, cones) populated in the environment and the *Insert* tab for managing objects (e.g., adding, removing). The right window displays the running environment where a SAU is represented as a cylinder shape (in the center) with two wheels each having its own actuator enabling differential drive. SAUs run autonomously in the environment using LiDAR streams to detect and avoid surrounding objects.

We created six independent running environments containing a combination of indoor, urban, and rural scenes. Each environment contains one SAU and 35 object types of which 15 object types are already known to the SAU and the rest are unknown. Object types vary in each environment. The six SAUs are labeled SAU1 to SAU6. Table 1 shows the object types that are known to SAU1 and the number of object instances considered in the environment. Other environments and SAUs are similar. We initially trained the system by running the six SAUs for 30 h to collect sufficient data for validation. After training, the experiment was run for 60 hours.

We used RVIZ, a 3-D visualizer for displaying data captured by sensors. Figure 6 shows the camera view displayed by RVIZ. Images furnished by RVIZ contain no noise (e.g., distortion, blurring), which enables B-kNN to make precise decisions. We used only images as input to B-kNN and Gazebo guarantees no failure of feeding images. Thus, there are no missing values for B-kNN attributes. We use k=1 for B-kNN to determine the first nearest neighbors for the target object. The

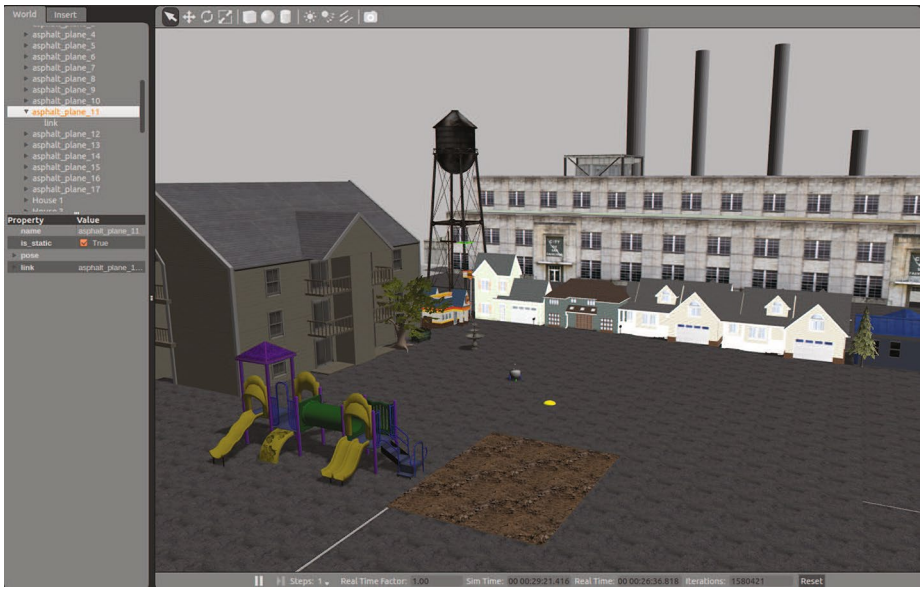

Fig. 5 Validation environment in Gazebo

Table 1 Object types known to SAU1

Num.	Object types	Num. of instances
1	Cube	100
2	Car	100
3	Cone	100
4	Bus	100
5	Train	100
6	TV	100
7	Dumpster	100
8	Gas pump	100
9	Table	100
10	Sofa	100
11	Chair	100
12	Fridge	100
13	Printer	100
14	Vending machine	100
15	Lockers	100
	Average	100

environmental characteristics (e.g., trees, walls) captured in an image are used to determine the type of the environment (e.g., indoor, urban, rural).

To demonstrate how a SAU avoids an object, Fig. 7 shows the SAU approaching a pedestrian, Fig. 8 shows the SAU changing the direction to the right as there is another pedestrian on the left, and Figs. 9, 10 illustrates the SAU successfully avoiding the pedestrians.

Fig. 6 RVIZ camera capture

Fig. 7 Gazebo SAU approaching pedestrians

Scenarios

We considered five scenarios for validation.

S1 Identifying an object known to LKB.

S2 Identifying an object unknown to LKB, but known to CKB. When an object is encountered, the SAU first identifies the type of the current environment (e.g., urban, rural). For an unknown object, the SAU sends via a web service an identification request to the CKB that is specialized for the environment type.

S3 Identifying an object unknown to LKB and CKB, but known to GKB. For an object unknown to the LKB and CKB, the CKB sends an identification request to the GKB. The GKB identifies the object and sends the existing adaptation plan back to the

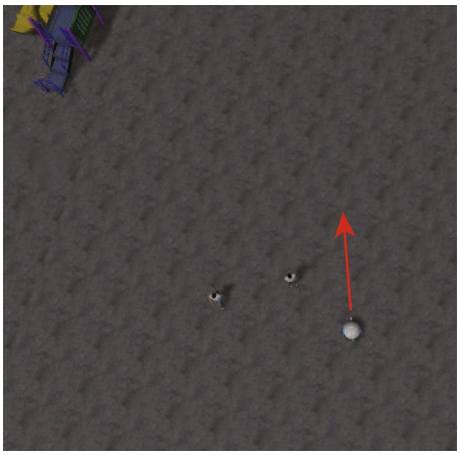

Fig. 8 Gazebo SAU changing direction

Fig. 9 Gazebo SAU successfully avoiding the pedestrians

Fig. 10 Gazebo SAU successfully avoided the pedestrians

CKB. The CKB updates its own dataset with the received plan and delegates the response to the requesting SAU.

S4 Identifying an object unknown to LKB, CKB, and GKB. The GKB creates a new type for the object and associates an adaptation plan to the object type.

S5 Failure of the adaptation plan provided by GKB. The SAU requests an alternative plan to the CKB which delegates the request to the GKB. The GKB builds an alternative plan and sends it back to the CKB. The CKB updates its own dataset with the alternative plan and delegates it to the SAU.

Scenario 1: Identifying objects known to LKB

This scenario evaluates the case where an SAU encounters an object that is known to its LKB. When the object is encountered, the SAU first determines the type of the current environment and tries to identify the object among the objects that have been previously identified in the respective environment. For a known object, the LKB already has an established adaptation plan. The plan is retrieved by the analyzer of the SAU and executed in consideration of the current speed and direction of the SAU.

Table 2 shows the number of objects encountered, recognized, and unrecognized by the LKB of the six SAUs in their respective environments without using CKBs or the GKB. The table shows that SAU1 had encountered 6452 objects, of which 3291 objects (51%) were recognized and 3161 objects (49%) were not recognized by the LKB. The average recognition rate of the six SAUs is 52% which reveals a significant limitation of complete reliance on LKBs. The table also shows the average response time to recognize objects for each SAU is measured as 0.12 s which is sufficiently shorter than the average collision time of 5 s (based on the average distance of 15 m to detect an object with the maximum speed of 3 m/s). This ensures that a response is received before collision.

Scenario 2: Identifying objects unknown to LKB, but known to CKB

This scenario evaluates identifying an object that is unknown to the LKB, but known to the CKB. For an object unknown to the LKB, the SAU sends an identification request via a web service to the CKB that is specialized for the type of the current environment where the object is encountered. If the CKB recognizes the object, it retrieves the existing adaptation plan and sends it back to the SAU. In the experiment, CKBs are initially furnished with 150 known objects. Table 3 shows the results of Scenario 2. It shows that

Table 2 The results of Scenarios 1

	Num. of encountered objects	Num. of recognized objects	Num. of unrecognized objects	Average response time (s)
SAU1	6452	3291 (51%)	3161 (49%)	0.10
SAU2	6587	3425 (52%)	3162 (48%)	0.12
SAU3	6210	3229 (52%)	2981 (48%)	0.13
SAU4	5958	3098 (52%)	2860 (48%)	0.12
SAU5	6741	3573 (53%)	3168 (47%)	0.14
SAU6	6367	3311 (52%)	3056 (48%)	0.13
Average	6386	3321 (52%)	3065 (48%)	0.12

Table 3 The results of Scenario 2

	Num. of encountered objects	Num. of recognized objects	Num. of unrecognized objects	Average response time (s)
SAU1	6452	4516 (70%)	1936 (30%)	0.21
SAU2	6587	4743 (72%)	1844 (28%)	0.24
SAU3	6210	4471 (72%)	1739 (28%)	0.21
SAU4	5958	4349 (73%)	1609 (27%)	0.22
SAU5	6741	4921 (73%)	1820 (27%)	0.22
SAU6	6367	4584 (72%)	1783 (28%)	0.23
Average	6386	4597 (72%)	1788 (28%)	0.22

on average, 72% of encountered objects are recognized and 28% are unrecognized, which demonstrates 20% improvement over the results of Scenario 1 in Table 2. The average response time is measured as 0.22 s which is sufficiently shorter than the average collision time.

Scenario 3: Identifying objects unknown to LKB and CKB, but known to GKB

This scenario evaluates identifying an object that is unknown to the LKB and CKB, but known to the GKB. For an object that is unknown to the CKB, the CKB sends an identification request to the GKB. The GKB then re-evaluates the environment of the object. If it is evaluated as the same type as the one that was initially evaluated by the SAU, the GKB starts to identify the object. For a known object, the GKB sends the existing adaptation plan back to the CKB and the CKB delegates the plan to the SAU. If the environment type is determined to be different from the one that was initially assessed by the SAU, the GKB delegates the request to the CKB that is specialized for the reassessed environment type. If the object is recognized by the CKB, the CKB responds to the SAU with the existing adaptation plan. If not, the CKB returns the request to the GKB. In the experiment, the GKB is initially furnished with 180 known objects out of 210 objects in the entire system.

Table 4 shows the results of Scenario 3. On average, 86% of encountered objects are recognized and 14% are unrecognized, which demonstrates 14% improvement over the results of Scenario 2 in Table 3. The average response time is measured as 0.39 s which remains sufficiently shorter than the average collision time.

Table 4 The results of Scenario 3

	Num. of encountered objects	Num. of recognized objects	Num. of unrecognized objects	Average response time (s)
SAU1	6452	5613 (87%)	839 (13%)	0.38
SAU2	6587	5599 (85%)	988 (15%)	0.41
SAU3	6210	5465 (88%)	745 (12%)	0.39
SAU4	5958	5303 (89%)	655 (11%)	0.39
SAU5	6741	5797 (86%)	944 (14%)	0.40
SAU6	6367	5412 (85%)	955 (15%)	0.38
Average	6386	5531 (86%)	854 (14%)	0.39

Table 5 The results of alternative Scenario 3

	Num. of encountered objects	Num. of recognized objects	Num. of unrecognized objects	Average response time (s)
SAU1	6452	5600 (87%)	852 (13%)	0.40
SAU2	6587	5619 (85%)	968 (15%)	0.42
SAU3	6210	5372 (87%)	838 (13%)	0.41
SAU4	5958	5035 (85%)	923 (15%)	0.42
SAU5	6741	5871 (87%)	870 (13%)	0.43
SAU6	6367	5380 (84%)	987 (16%)	0.42
Average	6386	5479 (86%)	906 (14%)	0.42

Table 6 Object type creation for unrecognized objects

	Num. of unrecognized objects	Num. of object types created	Num. of recognized objects	Average response time (s)
SAU1	839	4	428 (51%)	0.86
SAU2	988	3	524 (53%)	0.87
SAU3	745	5	380 (51%)	0.86
SAU4	655	7	334 (51%)	0.87
SAU5	944	5	491 (52%)	0.88
SAU6	955	4	506 (53%)	0.89
Average	854	5	444 (53%)	0.87

For an object known to the GKB, the GKB does not assess the environment type and tries to identify the object by itself without referring to another CKB. Table 5 show the results. The number of the recognized and unrecognized objects is similar to that in Scenario 3. However, the average response time is measured as 0.42 s which is slightly higher than that (0.39) of Scenario 3. This is because the GKB has to deal with a lager problem space.

Scenario 4: Identifying objects unknown to LKB, CKB, and GKB

This scenario evaluates the case in which the encountered object is unknown to the LKB, CKB and GKB. For such an object, the GKB creates a new object type using k-means clustering and establishes an adaptation plan for the new type using B-kNN. The GKB then sends the adaptation plan to the CKB and the CKB delegates the response to the SAU after updating its own dataset with the adaptation plan. Table 6 shows the number of object types created by the GKB for unrecognized objects. The table shows that SAU1 encountered 839 unrecognized objects, which results in four new object types created by the GKB. Using the new types, 428 objects (51%) of 839 unrecognized objects were recognized. The average response time including the creation time is measured as 0.86 s. Data on other SAUs can be interpreted similarly. On average, five new object types were created which enables identifying 444 objects (53%) of 854 unrecognized objects. The average response time is measured as 0.87 s which is sufficiently shorter than the average collision time.

Table 7 The results of Scenario 4

	Num. of encountered objects	Num. of recognized objects	Num. of unrecognized objects	Average response time (s)
SAU1	6452	6258 (97%)	194 (3%)	0.78
SAU2	6587	6284 (95%)	303 (5%)	0.81
SAU3	6210	5900 (95%)	311 (5%)	0.79
SAU4	5958	5720 (96%)	238 (4%)	0.77
SAU5	6741	6539 (97%)	202 (3%)	0.80
SAU6	6367	6112 (96%)	255 (4%)	0.79
Average	6386	6135 (96%)	250 (4%)	0.79

Table 7 shows the results of Scenario 4. The results show that SAU1 encountered 6452 encountered objects of which 6258 objects (97%) were identified and 194 objects (3%) were not recognized. This demonstrates 10% improvement over the results of Scenario 3 in Table 4. An object is marked as unidentified when the time before collision is reached before a viable adaptation plan is established. The average response time for recognizing an object in Scenario 4 is measured as 0.79 s. Note that this is shorter than the time in Scenario 3 (see Table 4). This is because the average response time (0.12 s) of LKB is shorter than the average response time (0.39 s) of GKB. Data on other SAUs can be interpreted similarly. On average, of 6386 encountered objects, 6135 objects (96%) were successfully recognized and 250 objects (4%) were not recognized. The average of average response times is measured as 0.79 s which is sufficiently shorter than the average collision time.

Scenario 5: Sending Feedback to GKB

If the adaptation plan received from the GKB fails (e.g., causing a near collision), the SAU requests an alternative plan to the GKB. Such a request is referred to as feedback. Upon receiving a feedback, the GKB searches for a k near object using B-kNN and sends the adaptation plan associated with the object as an alternative plan. If no object is found at the k distance, the GKB searches for another object in the next $k+1$ distance. This continues until either the plan is successful or the minimum time before collision has reached, whichever comes first. Table 8 shows the results of Scenario 5. The tables show that SAU1 had encountered 6452 objects of which 710 needed feedbacks to be identified and 1859 feedbacks were sent to the GKB. There were 7602 attempts for identifying

Table 8 The results of Scenario 5

	Num. of encountered objects	Num. of objects needed feedback	Num. of attempts	Num. of feedback	Num. of successfully executed feedbacks	Avg. resp. time (s)
SAU1	6452	710	7602	1859	516	0.98
SAU2	6587	788	7698	1899	485	0.91
SAU3	6210	487	6945	1222	176	0.98
SAU4	5958	471	6590	1103	233	0.99
SAU5	6741	816	7997	2072	613	0.90
SAU6	6367	798	7469	1900	544	0.97
Average	6386	678	7383	1676	428	0.95

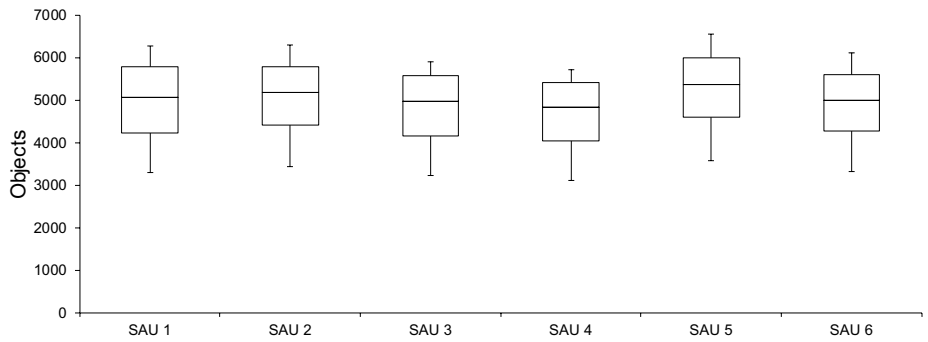

Fig. 11 Comparing results of Scenario 1–4

Table 9 Confusion matrix for SAU1

	Object types identified by B-kNN									
	Cube	**Car**	**Cone**	**Bus**	**Train**	**TV**	**Dumpster**	**Gas pump**	**Table**	**Sofa**
Actual object types										
Cube	100	0	0	0	0	3	0	0	3	0
Car	0	100	0	4	5	0	0	0	0	0
Cone	2	0	100	0	0	0	0	0	0	0
Bus	0	3	0	98	6	0	0	0	0	0
Train	0	1	0	4	96	0	3	0	0	0
TV	2	0	0	0	0	100	0	0	0	0
Dumpster	0	0	0	0	0	0	99	0	0	0
Gas Pump	0	0	2	0	0	0	0	100	0	0
Table	2	0	0	0	0	0	0	0	99	0
Sofa	0	0	0	0	0	0	0	0	0	100

objects of which 516 were successfully executed. Data on other SAUs can be interpreted similarly. The average response time for both successful adaptations and feedbacks is 0.95 s which is sufficiently shorter than the average collision time. The average number of feedbacks per object is 2.84 (1396/492).

Figure 11 shows the average number of recognized objects for Scenario 1–4. The figure shows that the number of average recognized objects is increased from 3321 (52%) in Scenario 1 up to 6135 (96%) in Scenario 4.

Quantitative analysis

We analyze the approach in terms of accuracy, precision, and recall using the confusion matrix (Fawcett 2006). Table 9 shows a partial confusion matrix for the performance of SAU1 without using CKBs or the GKB. Columns represent the object types identified by B-kNN and rows represent actual object types. For instance, for the TV type, two objects were identified as cube by B-kNN, but they were actually TVs.

Based on the confusion matrix, we measure the accuracy, precision, and recall of the approach. We first measured true positive (TP), true negative (TN), false positive (FP), and false negative (FN) as shown in Table 10. For the cube type, (1) TP is measured as 100 which represents the number of objects that were correctly identified as cube by

Table 10 TP, FP, TN, and FN

Object types	TP	FP	TN	FN
Cube	100	11	2930	9
Car	100	4	2937	9
Cone	100	2	2946	2
Bus	98	8	2937	9
Train	96	11	2939	8
TV	100	5	2941	4
Dumpster	99	3	2947	2
Gas pump	100	4	2944	2
Table	99	4	2946	2
Sofa	100	4	2938	8
Chair	89	8	2956	8
Fridge	99	9	2936	7
Printer	100	5	2941	4
Vending machine	100	8	2931	11
Lockers	100	9	2931	10

B-kNN, (2) FP is measured as 11 which represents the number of objects that were incorrectly identified as cube by B-kNN, but they are actually non-cube objects, (3) TN is measured as 2930 which represents the number of objects that were correctly identified as non-cube by B-kNN, and (4) FN is measured as 9 which represents the number of objects that were incorrectly identified as non-cube by B-kNN, but are actually cubes. Other data in the table can be interpreted similarly.

Based on the values in Table 10, the accuracy, precision, recall, and F1 score of the approach are calculated using the following formulas.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

Table 11 shows that the average of accuracy, recall, precision, recall, and F1 are measured as 99.58%, 94.01%, 94.04%, and 94.01% respectively.

We also compare this work with our previous work which uses the traditional kNN in terms of object recognition and response time. Table 12 shows the results. The results demonstrate 43% improvement in response time while having the same accuracy in object recognition, which justifies the adoption of B-kNN in this work.

Table 11 Accuracy, precision, and recall

Object types	Accuracy (%)	Recall (%)	Precision (%)	F1 score (%)
Cube	99.34	90.09	91.74	90.91
Car	99.57	96.15	91.74	93.90
Cone	99.87	98.04	98.04	98.04
Bus	99.44	92.45	91.59	92.02
Train	99.38	89.72	92.31	91.00
TV	99.70	95.24	96.15	95.69
Dumpster	99.84	97.06	98.02	97.54
Gas pump	99.80	96.15	98.04	97.09
Table	99.80	96.12	98.02	97.06
Sofa	99.61	96.15	92.59	94.34
Chair	99.48	91.75	91.75	91.75
Fridge	99.48	91.67	93.40	92.52
Printer	99.70	95.24	96.15	95.69
Vending machine	99.38	92.59	90.09	91.32
Lockers	99.38	91.74	90.91	91.32
Average	99.58	94.01	94.04	94.01

Table 12 Improvement in response time

	% of recognized objects	Num. of encountered objects	Num. of recognized objects	Response time (s)
Presented approach	96	38315	36813	0.79
Previous approach	96	14963	14435	1.56

Table 13 Approach validation

	Num. of SAUs	Num. of encountered objects	Num. of identified objects	Num. of unidentified objects	Average response time
Presented approach	6	38315	36813 (96%)	1502 (4%)	0.79
Traditional approach	6	37854	19315 (51%)	18539 (49%)	0.13
Cloud approach	6	37718	36239 (96%)	1479 (4%)	1.24

Comparative analysis

We also compared the presented approach with the traditional approach relying on LKBs only and the cloud-based approach relying on CKBs and the GKB only. We simulated the traditional approach by running six independent SAUs simultaneously, without using CKBs and the GKB. To simulate the cloud-based approach, we configured the system to run six SAUs using only CKBs and the GKB. In all the three approaches, SAUs were run for the same duration of 100 h. Table 13 shows the results of the comparison. The results show that in the presented approach, 38315 objects were encountered, of which 36813 (96%) were identified, leaving only 1502 (4%) unidentified with the average response time of 0.79 s. The traditional approach encountered 37854 objects of which 19315 (51%) were identified with the average response time of 0.13 s. This demonstrates 45% improvement in object identification with an increase of 0.66 s loss in the average

response time due to communication overheads. The improvement mainly results from the maturity of CKBs and the GKB which are built upon more than 45,000 records collected from all SAUs sharing CKBs and the GKB. The improvement is also attributed to the fact that the objects used in the experiment are relatively simpler and easier to recognize as the work focuses on object recognition not image processing. The traditional approach shows similar results to those of Scenario 1 in Table 2 as they both use only LKBs.

In the cloud-based approach, 37718 objects were encountered of which 36239 (96%) were identified with the average response time of 1.74 s due to communication overheads. Compared to the cloud-based approach, the presented work has the same precision in object identification and 0.95 s less response time. The time saving mainly results from the use of LKB before requesting to the CKB and the GKB.

Conclusion

In this work, we have presented a three-phase decision making approach for self-adaptive vehicle systems to improve the precision and performance in object identification with competitive precision. To improve precision, the approach makes use of both the local knowledge base in the SAU and the context-specific and global knowledge bases through web services. To improve performance, the approach uses B-kNN. The validation demonstrates that the presented approach outperforms the traditional approach by 45% improvement in object identification and the cloud-based approach by 0.95 s in average response time. On the other hand, the presented approach takes 0.66 s more in average response time over the traditional approach, while remaining competitive in precision over the cloud-based approach.

The presented three-phase decision making approach can be applied to other domains such as the computer network domain for smart routing and the smart grid domain for intelligent electrical devices (IEDs) where self-adaptability can be utilized. We plan to investigate an extend to which the presented approach can be applied to the automotive domain where vehicles can share their knowledge base with neighboring vehicles through vehicle-to-vehicle (V2V) communication. This can be viewed as sharing LKBs of SUAs, which is different from sharing CKB and GKB. For such sharing in V2V communication, accessibility as to who can share and types of data to be shared should be addressed with timing constraints which are critical in a real-time domain.

Authors' contributions

All Authors have equal contributions to this research. All authors read and approved the final manuscript.

Authors' information

Dhrgam AL Kafaf is a research engineer and lecturer. He received the Ph.D. in Computer Science from Oakland University, M.S. in Computer Science from Lawrence Technological University, and he received B.S.E. in Computer and Software Engineering from University of Technology Baghdad in 2018, 2011 and 2002 respectively. His research interests include machine learning, self-adaptive systems, and autonomous vehicles.

Dae-Kyoo Kim is an associate professor of the Department of Computer Science and Engineering at Oakland University. He received the Ph.D. in Computer Science from Colorado State University in 2004. During his Ph.D. work, he worked as a technical specialist at NASA\Ames Research Center in 2003. He is a senior member of the IEEE Computer Society.

Lunjin Lu is an associate professor in the Computer Science and Engineering Department at Oakland University. His broad research areas are programming languages and software engineering. His research interests include semantic-based program analysis, program semantics, logic programming, UML modeling, software design patterns, software refinement, UML model refinement, and pattern conformance.

Acknowledgements
To Oakland University for supporting this research and for providing the labs to conduct the experiments for this research.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
The datasets generated during and/or analysed during the current study are not publicly available. but are available from the corresponding author on reasonable request.

Funding
The software Engineering lab at Oakland University funding this research.

Publisher's Note

Received: 11 August 2018 Accepted: 8 October 2018
Published online: 23 October 2018

References

Bleicher M, Zabrodin E, Spieles C, Bass SA, Ernst C, Soff S, Bravina L, Belkacem M, Weber H, Stöcker H, Greiner W (1999) Relativistic hadron-hadron collisions in the ultra-relativistic quantum molecular dynamics model. J Phys G Nucl Part Phys 25(9):1859

Bonaccorsi M, Fiorini L, Sathyakeerthy S, Saffiotti A, Cavallo F, Dario P (2015) Design of cloud robotic services for senior citizens to improve independent living in multiple environments. Intell Artif 9(1):63–72

Bradski G, Kaehler A (2008) Learning OpenCV: computer vision with the OpenCV Library. O'Reilly Media, Inc.

Chen Y, Du Z, García-Acosta M (2010) Robot as a service in cloud computing. In: Proceedings of the 5th IEEE international symposium on service oriented system engineering (SOSE), pp 151–158

Cheng SW, Garlan D, Schmerl B (2005) Making self-adaptation an engineering reality. In: Proceedings of self-star properties in complex information systems, pp 158–173

Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13(1):21–27

Dorigo M, Schnepf U (1993) Genetics-based machine learning and behavior-based robotics: a new synthesis. IEEE Trans Syst Man Cybern 23(1):141–154

Fawcett T (2006) An introduction to ROC analysis. Pattern Recogn Lett 27(8):861–874

Garlan D, Cheng S-W, Schmerl B (2003) Increasing system dependability through architecture-based self-repair. In: de Lemos R, Gacek C, Romanovsky A (eds) Architecting dependable systems. Springer, Berlin, pp 61–89

Grubbs FE (1969) Procedures for detecting outlying observations in samples. Technometrics 11(1):1–21

Haug EJ (1989) Computer aided kinematics and dynamics of mechanical systems. Allyn and Bacon, Boston

Hickman R, Kuffner JJ Jr, Bruce J.R, Gharpure C, Kohler D, Poursohi A, Francis AG Jr, Lewis T (2014) Shared robot knowledge base for use with cloud computing system. US Patent 8,639,644

Hu G, Tay WP, Wen Y (2012) Cloud robotics: architecture, challenges and applications. IEEE Netw 26(3):21–28

Kafaf DA, Kim D-K (2017) A web service-based approach for developing self-adaptive systems. Comput Electr Eng 63:260–276

Kafaf DAL, Kim D-K, Lu L (2017) B-knn to improve the efficiency of kNN. In: Proceedings of the 6th international conference on data science, technology and applications. Science and Technology Publications, pp 126–132

Kehoe B, Matsukawa A, Candido S, Kuffner J, Goldberg K (2013) Cloud-based robot grasping with the google object recognition engine. In: Robotics and automation (ICRA), 2013 IEEE international conference on, IEEE, pp 4263–4270

Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50

Koenig N, Howard A (2004) Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proceedings of IEEE/RSJ international conference on intelligent robots and systems, vol 3, pp 2149–2154

Kramer J, Magee J (2007) Self-managed systems: an architectural challenge. In: Proceedings of future of software engineering, pp 259–268

Liu B, Chen Y, Blasch E, Pham K, Shen D, Chen G (2014) A holistic cloud-enabled robotics system for real-time video tracking application. In: Proceedings of future information technology, pp 455–468

Magee J, Kramer J (1996) Dynamic structure in software architectures. In: ACM SIGSOFT software engineering notes, vol 21, issuse 6, pp 3–14

Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: an open-source robot operating system. In: Proceedings of ICRA workshop on open Source software

Richardson L, Ruby S (2008) RESTful web services. O'Reilly Media, Inc.

Tian G, Chen H, Lu F (2015) Cloud computing platform based on intelligent space for service robot. In: Information and automation, 2015 IEEE international conference on, pp 1562–1566

Wang Y, de Silva CW (2008) A machine-learning approach to multi-robot coordination. Eng Appl Artif Intell 21(3):470–484