frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 25 May 2016
doi: 10.3389/frobt.2016.00026

# A Modular Software Framework for Eye–Hand Coordination in Humanoid Robots

*Jürgen Leitner[1]*, Simon Harding[2], Alexander Förster[3] and Peter Corke[1]*

[1] Australian Centre for Robotic Vision, Queensland University of Technology, Brisbane, QLD, Australia, [2] Machine Intelligence Ltd., South Zeal, UK, [3] Institute for Artificial Intelligence, Universität Bremen, Bremen, Germany

We describe our software system enabling a tight integration between vision and control modules on complex, high-DOF humanoid robots. This is demonstrated with the *iCub* humanoid robot performing visual object detection and reaching and grasping actions. A key capability of this system is reactive avoidance of obstacle objects detected from the video stream while carrying out reach-and-grasp tasks. The subsystems of our architecture can independently be improved and updated, for example, we show that by using machine learning techniques we can improve visual perception by collecting images during the robot's interaction with the environment. We describe the task and software design constraints that led to the layered modular system architecture.

**Keywords: humanoid robots, software framework, robotic vision, eye–hand coordination, reactive reaching, machine learning**

OPEN ACCESS

***Edited by:***
*Lorenzo Natale,*
*Istituto Italiano di Tecnologia, Italy*

***Reviewed by:***
*Matthias Rolf,*
*Oxford Brookes University, UK*
*Nikolaus Vahrenkamp,*
*Karlsruhe Institute of Technology, Germany*
*Giovanni Saponaro,*
*Instituto Superior Técnico, Portugal*

****Correspondence:***
*Jürgen Leitner*
*j.leitner@roboticvision.org*

***Specialty section:***
*This article was submitted to Humanoid Robotics, a section of the journal Frontiers in Robotics and AI*

***Received:*** *29 November 2015*
***Accepted:*** *15 April 2016*
***Published:*** *25 May 2016*

***Citation:***
*Leitner J, Harding S, Förster A and Corke P (2016) A Modular Software Framework for Eye–Hand Coordination in Humanoid Robots. Front. Robot. AI 3:26. doi: 10.3389/frobt.2016.00026*

## 1. INTRODUCTION

In the last century, robots have transitioned from science fiction to science fact. When interacting with the world around them robots need to be able to reach for, grasp, and manipulate a wide range of objects in arbitrary positions. Object manipulation, as this is referred to in robotics, is a canonical problem for autonomous systems to become truly useful. We aim to overcome the limitations of current robots and the software systems that control them, with a focus on complex bi-manual robots. It has previously been suggested that better perception and coordination between sensing and acting are key requirements to increase the capabilities of current systems (Kragic and Vincze, 2009; Ambrose et al., 2012). Yet with the increasing complexity of the mechanical systems of modern robots programing these machines can be tedious, error prone, and inaccessible to non-experts. Roboticists are increasingly considering learning over time to "program" motions into robotic systems. In addition, continuous learning increased the flexibility and provides the means for self-adaptation, leading to more capable autonomous systems. Research in artificial intelligence (AI) techniques has led to computers that can play chess on a level good enough to win against (and/or tutor) the average human player (Sadikov et al., 2007). Robotic manipulation of chess pieces on a human-level of precision and adaptation is still beyond current systems.

The problem is not with the mechanical systems. Sensory feedback is of critical importance for acting in a purposeful manner. For humans particularly, vision is an important factor in the development of reaching and grasping skills (Berthier et al., 1996; McCarty et al., 2001). The essential challenge in robotics is to create a similarly efficient perception system. For example, NASA's Space Technology Roadmap is calling for the development of autonomously calibrating hand-eye systems enabling successful off-world robotic manipulation (Ambrose et al., 2012). This ability is fundamental for

humans and animals alike, leading to many experimental studies on how we perform these actions (Posner, 1989; Jeannerod, 1997). The process is still not fully understood but basic computational models for how humans develop their reaching and grasping skills during infancy exist (Oztop et al., 2004). Where 14-month-old infants can imitate and perform simple manipulation skills (Meltzoff, 1988), robots can only perform simple, pre-programed reaching and grasping in limited scenarios. Our ability to adapt during motion execution to changing environments is lacking in robots right now. Yet this adaptation is important as even if the environment can be perceived precisely, it will not be static in most (interesting) settings.

Coming back to the chess example, for an autonomous system to pick up a chess piece, it needs to be able to perceive the board, detect the right piece, and locate the position accurately, before executing a purposeful motion that is safe for the robot and its environment. These sub-problems have turned out to be much harder than expected a few decades ago. With the progress in mechanical design, motion control, and computer vision, it is time to revisit the close coupling between those systems to create robots that perform actions in day-to-day environments.

## 1.1. Motion and Action: Interacting with the Environment

In the chess example, even if the state of the board and its location are known perfectly, moving a certain chess piece from one square to another without toppling other pieces is a non-trivial problem. Children, even at a very young age, have significantly better (more "natural," smoother) hand movements than almost all currently available humanoid robots. In humans, the development of hand control starts at an early age, albeit clumsily, and the precision grasp is not matured until the age of 8–10 years (Forssberg et al., 1991). Even after manipulation skills have been learnt, they are constantly adapted by a perception–action loop to yield desired results during action execution. Vision and action are closely integrated in the human brain. Various specializations develop also in the visual pathways of infants related to extracting and encoding information about the location and graspability of objects (Johnson and Munakata, 2005).

To enable robots to interact with objects in unstructured, cluttered environments, a variety of reactive approaches have been investigated. These quickly generate control commands based on sensory input – similar to reflexes – without sampling the robot's configuration space and deliberately searching for a solution (Khatib, 1986; Brooks, 1991; Schoner and Dose, 1992). Generally such approaches apply a heuristic to transform local information (in the sensor reference frame) to commands sent to the motors, leading to fast, reflex-like obstacle avoidance. Reactive approaches have become popular in the context of safety and human-robot interaction (De Santis et al., 2007; Dietrich et al., 2011) but are brittle and inefficient at achieving global goals. A detailed model of the world enables the planning of coordinated actions. Finding a path or trajectory is referred to as the path planning problem. This search for non-colliding poses is generally expensive and increasingly so with higher DOF. Robots controlled this way are typically slow and appear cautious in their motion execution. These reactive approaches started to appear in the 1980s as an alternative to the "think first, act later" paradigm. Current robotic systems operate in a very sequential manner. After a trajectory is planned, it is performed by the robot, before the actual manipulation begins. We aim to move away from such brittle global planner paradigms and have these parts overlap and have continuous refining based on visual feedback. The framework provides quick and reactive motions, as well as, interfaces to these for so higher-level agents or "opportunistic" planners can control the robot safely.

Once the robot has moved its end-effector close enough to an object, it can start to interact with it. In recent years, good progress has been made in this area thanks to the development of robust and precise grippers and hands and the improvement of grasping techniques. In addition, novel concepts of "grippers" appeared in research, including some quite ingenious solutions, such as the granular gripper (Brown et al., 2010). As alternative to designing a grasping strategy, it may be possible to learn it using only a small number of real-world examples, where good grasping points are known, and these could be generalized or transferred to a wide variety of previously unseen objects (Saxena et al., 2008). An overview of the state of research in robot grasping is found in Carbone (2013). Our framework provides an interface to "action repertoires." In one of the examples later on, we show how we use a simple grasping module that is triggered when the robot's end-effector is close to a target object. While vision may be suitable for guiding a robot to an object, the very last phase of object manipulation – the transition to contact – may require the use of sensed forces.

## 1.2. Robotic Vision: Perceiving the Environment

For a robot to pick a chess piece, for example, finding the chess board and each of the chess pieces in the camera image or even just to realize that there is a chess board and pieces in the scene is critical. An important area of research is the development of artificial vision systems that provide robots with such capabilities. The robot's perception system needs to be able to determine whether the image data contain some specific object, feature, or activity. While closely related to computer vision, there are a few differences mainly in how the images are acquired and how the outcome will provide input for the robot to make informed decisions. For example, visual feedback has extensively been used in mobile robot applications for obstacle avoidance, mapping, and localization (Davison and Murray, 2002; Karlsson et al., 2005). Especially in the last decade, there has been a surge of computer vision research. A focus is put on the areas relevant for object manipulation[1] and the increased interest in working around and with humans.

Robots are required to detect objects in their surroundings even if they were previously unknown. In addition, we require them to be able to build models so they can re-identify

[1]In recent years, various challenges have emerged around this topic, such as the Amazon Picking Challenge and RoboCup@Home.

and memorize them in the future. Progress has been made on detecting objects – especially when limiting the focus on specific settings – the interested reader is referred to current surveys (Cipolla et al., 2010; Verschae and Ruiz-del Solar, 2015). A solution for the general case, i.e., detecting arbitrary objects in arbitrary situations, is elusive though (Kemp et al., 2007). Environmental factors, including changing light conditions, inconsistent sensing, or incomplete data acquisition seem to be the main cause of missed or erroneous detection (Kragic and Vincze, 2009) (see also the environmental changes in **Figure 1**). Most object detection applications have been using hand-crafted features, such as SIFT (Lowe, 1999) or SURF (Bay et al., 2006), or extensions of these for higher robustness (Stückler et al., 2013). Experimental robotics still relies heavily on artificial landmarks to simplify (and speed-up) the detection problem, though there is recent progress specifically for the *iCub* platform (Ciliberto et al., 2011; Fanello et al., 2013; Gori et al., 2013). Many AI and learning techniques have been applied to object detection and classification over the past years. Deep-learning has emerged as a promising technology for extracting general features from ever larger datasets (LeCun et al., 2015; Schmidhuber, 2015). An interface to such methods is integrated in our framework and has been applied to autonomously learn object detectors from small datasets (only 5–20 images) (Leitner et al., 2012a, 2013a; Harding et al., 2013).

Another problem relevant to eye–hand coordination is estimating the position of an object with respect to the robot and its end-effector. "Spatial Perception," as this is known, is a requirement for planning useful actions and build cohesive world models. Studies in brain- and neuro-science have uncovered trends on *what* changes, when we learn to reason about distances by interacting with the world, in contrast *how* these changes happen is not yet clear (Plumert and Spencer, 2007). In robotics, to obtain a distance measure multiple camera views will provide the required observations. Projective geometry and its implementation in stereo vision systems are quite common on robotic platforms. An overview of the theory and techniques can be found in Hartley and Zisserman (2000). While projective geometry approaches work well under carefully controlled experimental circumstances, they are not easily transferred to robotics applications though. These methods are falling short as there are either separately movable cameras (such as in the case of the *iCub*, which can be seen in the imprecise out-of-the-box localization module (Pattacini, 2011)) or only single cameras available (as with Baxter). In addition, the method needs to cope with separate movement of the robot's head, gaze, and upper body. A goal for the framework was also to enable the learning of depth estimation from separately controllable camera pairs, even on complex humanoid robots moving about (Leitner et al., 2012b).

## 1.3. Integration: Sensorimotor Coordination

Although there exists a rich body of literature in computer vision, path planning, and feedback control, wherein many critical sub-problems are addressed individually, most demonstrable behaviors for humanoid robots do not effectively integrate elements from all three disciplines. Consequently, tasks that seem trivial to humans, such as picking up a specific object in a cluttered environment, remain beyond the state-of-the-art in experimental robotics. A close integration of computer vision and control is of importance, e.g., it was shown that to enable a 5 DOF robotic arm to pick up objects just providing a point-cloud generated model of the world was not sufficient to calculate reach and grasp behaviors on-the fly (Saxena et al., 2008). The previously mentioned work by Maitin-Shepard et al. (2010) was successful, manipulating towels due to a sequence of visually guided re-grasps. "Robotics, Vision, and Control" (Corke, 2011) puts the close integration of these components into the spotlight and describes common pitfalls and issues when trying to build systems with high levels of sensorimotor integration.

Visual Servoing (Chaumette and Hutchinson, 2006) is a commonly used approach to create a tight coupling of visual perception and motor control. The closed-loop vision-based control can be seen as a very basic level of eye–hand coordination. It has been shown to work as a functional strategy to control robots without any prior calibration of camera to end-effector transformation (Vahrenkamp et al., 2008). A drawback of visual servoing is that it requires the robust extraction of visual features; in addition,

**FIGURE 1 | During a stereotypical manipulation task, object detection is a hard but critical problem to solve**. These images collected during our experiments show the changes in lighting, occlusions, and pose of complex objects. (Note: best viewed in color) We provide a framework that allows for the easy integration of multiple, new detectors (Leitner et al., 2013a).

the final configuration of these features in image space needs to be known *a priori*.

Active vision investigates how controlling the motion of the camera, i.e., where to look at, can be used to create additional information from a scene. (Welke et al., 2010), for example, presented a method that creates a segmentation out of multiple viewpoints of an object. These are generated by rotating an object in the robot's hand in front of its camera. These exploratory behaviors are important to create a fully functioning autonomous object classification system and are highlighting one of the big differences between computer and robotic vision.

Creating a system that can improve actions by using visual feedback, and vice versa improve visual perception by performing manipulation actions, necessitates a flexible way of representing, learning, and storing visual object descriptions. We have developed a software framework for creating a functioning eye–hand coordination system on a humanoid robot. It covers quite distinct areas of robotics research, namely machine learning, computer vision, and motion generation. Herein, we describe and showcase this modular architecture that combines those areas into an integrated system running on a real robotic platform. It was started as a tool for *iCub* humanoid but thanks to its modular design it can and has been used with other robots, most recently on a *Baxter* robot as well.

### 1.3.1. Robotic Systems Software Design and Toolkits

Current humanoid robots are stunning feats of engineering. With the increased complexity of these systems, the software to run these machines is increasing in complexity as well. In fact, programing today's robots requires a big effort and usually a team of researchers. To reduce the time needed to setup robotics experiments and to stop the need to repeatedly invent the wheel, good system level tools are needed. This has led to the emergence of many open source projects in robotics (Gerkey et al., 2003; van den Bergen, 2004; Metta et al., 2006; Jackson, 2007; Diankov and Kuffner, 2008; Fitzpatrick et al., 2008; Quigley et al., 2009). State-of-the art software development methods have also been translated into the robotics domain. Innovative ideas have been introduced in various areas to promote the reuse of robotic software "artifacts," such as components, frameworks, and architectural styles (Brugali, 2007). To build more general models of robot control, robotic vision and their close integration robot software needs to be able to abstract certain specificities of the underlying robotic system. There exists a wide variety of middleware systems that abstract the specifics of each robot's sensors and actuators. Furthermore, such systems need to provide the ability to communicate between modules running in parallel on separate computers.

ROS (Robot Operating System) (Quigley et al., 2009) is one of the most popular robotics software platforms. At heart, it is a component-based middleware that allows computational nodes to publish and subscribe to messages on particular topics, and to provide services to each other. Nodes communicate via "messages," i.e., data blocks of pre-defined structure, and can execute a networked distributed computer system and the connections can be changed dynamically during runtime. ROS also contains a wider set of tools for computer vision (OpenCV and point-cloud library PCL), motion planning, visualization, data logging and replay, debugging, system startup as well as drivers for a multitude of sensors, and robot platforms. For the *iCub* YARP (Yet Another Robotics Platform) (Metta et al., 2006) is the middleware of choice. It is largely written in C++ and uses separately running code instances, titled "modules." These can be dynamically and flexibly linked and communicate via concise and pre-defined messages called "bottles," facilitating component-based design. There is a wide range of other robotic middleware systems available, such as ArmarX (Vahrenkamp et al., 2015), OROCOS (Soetens, 2006), and OpenRTM (Ando et al., 2008), all with their own benefits and drawbacks, see (Elkady and Sobh, 2012) for a comprehensive comparison.

The close integration of vision and control has been addressed by VISP (Visual Servoing Platform) developed at INRIA (Marchand et al., 2005). It provides a library for controlling robotic systems based on visual feedback. It contains a multitude of image processing operations, enabling robots to extract useful features from an image. By providing the desired feature values, a controller for the robot's motion can be derived (Hutchinson et al., 1996; Chaumette and Hutchinson, 2006). The framework presented here is building on these software systems to provide a module-based approach to tightly integrate computer vision and motion control for reaching and grasping on a humanoid robot. The architecture grew naturally over the last few years and was initially designed for the *iCub* and, hence, used YARP. While there exists also a "bridge" component in YARP allowing it to communicate with ROS topics and nodes, it was easy to port it to ROS and Baxter. Furthermore, there is currently a branch being developed aimed to be fully agnostic to the underlying middleware.

## 2. THE EYE–HAND FRAMEWORK

The goal of our research is to improve the autonomous skills of humanoid robots by providing a library giving a solid base of sensorimotor coordination. To do so, we developed a modular framework that allows to easily run and repeat experiments on humanoid robots. To create better perception and motion, as well as a coordination between those, we split the system into two subsystems: one focusing on action and the other one on vision (our primary sense). To deal with uncertainties, various machine learning (ML) and artificial intelligence (AI) techniques are applied to support both subsystems and their integration. We close the loop and perform grasping of objects, while adapting to unknown, complex environments based on visual feedback, showing that combining robot learning approaches with computer vision improves adaptivity and autonomy in robotic reaching and grasping.

Our framework, sketched in **Figure 2A**, provides an integrated system for eye–hand coordination. The Perception (green) and Action (yellow) subsystems are supported by Memory (in blue) that enables the persistent modeling of the world. Functionality has grown over time and the currently existing modules that have been used in support of eye–hand coordination framework for cognitive robotics research (Leitner, 2014, 2015) are as follows:

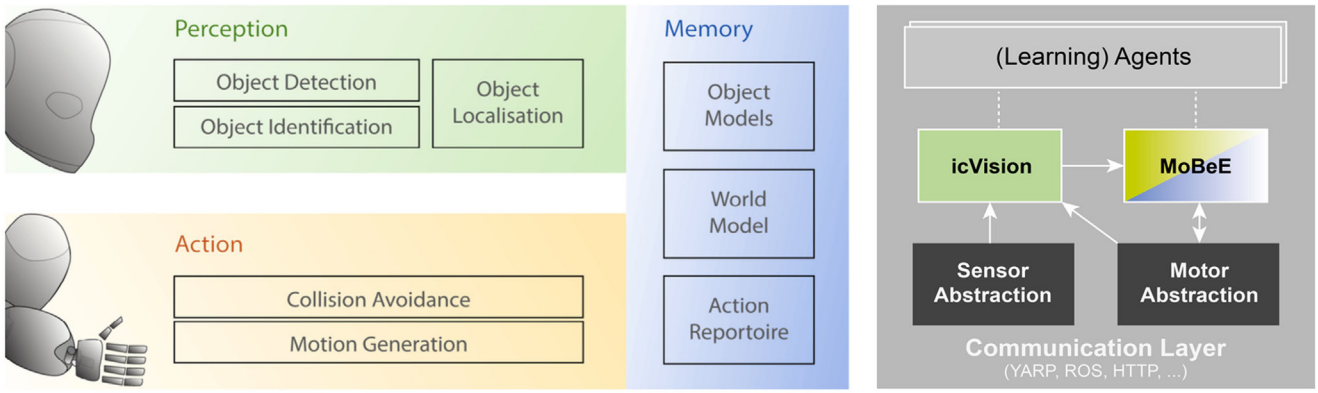


**FIGURE 2** | **(A)** Overview of the common subsystems for a functional eye–hand coordination on humanoid robots. In broader terms, one can separate the (visual) perception side (in green) from the action side (yellow). In addition to these, a memory subsystem (blue) allows to build-up an action repertoire and a set of object models. **(B)** The presented framework herein consists of a modular way of combining perception tasks, encapsulated in the *icVision* subsystem [green, as in **(A)**], with the action side and a world model, represented by the *MoBeE* subsystem (in yellow and blue). In addition, agents can interface these systems to generate specific behaviors or to learn from the interaction with the environment (see Results). To allow portability, the system uses a communication layer and a robot abstraction middleware, e.g., ROS or YARP.

- *Perception: Object Detection and Identification:* as mentioned above, the detection and identification of objects is a hard problem. To perform object detection and identification, we use a system called *icVision*. It provides a modular approach for the parallel execution of multiple object detectors and identifiers. While these can be hard-coded (e.g., optical flow segmentation of moving object, or simple color thresholding), the main advantage of this flexible system is that it can be interfaced by a learning agent (sketched in **Figure 2B**). In our case, we have successfully used Cartesian Genetic Programing for Image Processing (CGP-IP) (Harding et al., 2013) as an agent to learn visual object models in both a supervised and unsupervised fashion (Leitner et al., 2012a). The resulting modules perform specific object segmentation of the camera images.
- *Perception: Object Localization: icVision* also provides a module for estimating the location of an object detected by multiple cameras – i.e., the two eyes in the case of the *iCub*. In this case, again the flexibility of the perception framework allows for a learning agent to predict object positions with a technique based on genetic programing and an artificial neural network estimators (Leitner et al., 2012b). These modules can be easily swapped or run in parallel, even on different machines.
- *Action: Collision Avoidance and Motion Generation: MoBeE* is used to safeguard the robot from collisions both with itself and the objects detected. This is implemented as a low-level interface to the robot and uses virtual forces based on the robot kinematics to generate the robot's motion. A high-level agent or planner can provide the input to this system (more details in the next section).
- *Memory: World Model:* In addition to modeling the kinematics of the robot to *MoBeE* also keeps track of the detected object in operational space. It is also used as a visualization for the robot's current belief state by highlighting (impeding) collisions (see Section 2.2).
- *Memory: Action Repertoire:* a light-weight, easy-to-use, one-shot grasping system is used. It can be configured to perform a variety of grasps, all requiring to close the fingers in a coordinated fashion. The *iCub* incorporates touch sensors on the fingertips, but due to the high noise, we use the error reported by the PID controllers of the finger motors to know when they are in contact with the object.

Complex, state-of-the art humanoid robots are controlled by a distributed system of computers most of which are not onboard the robot. On the *iCub* (and similarly on *Baxter*), an umbilical provides power to the robot and a local-area-network (LAN) connection. **Figure 3** sketches the distributed computing system used to operate a typical humanoid robot: very limited on-board computing, which mainly focuses on the low-level control and sensing, is supported by networked computers for computational intensive tasks. The *iCub*, for example, employs an on-board PC104 controller that communicates with actuators and sensors using CANBus. Similarly, *Baxter* has an on-board computing system (Intel i7) acting as the gateway to joints and cameras. More robot-specific information about setup and configuration, as well as the code base, can be found on the *iCub* and *Baxter* Wiki pages,[2] where researchers, from a large collection of research labs using the robot, contribute and build up a knowledge base.

All the modules described communicate with each other using a middleware framework (depicted in **Figure 2B**). The first experiments were performed on the *iCub*; therefore, the first choice for the middleware was YARP. A benefit of using a robotic middleware is that actuators and sensors can be abstracted, i.e., the modules that connect to *icVision* and *MoBeE* do not require to know the robot specifics. Another benefit of building on existing robotics middleware is the ability to distribute modules across

[2] *iCub* Wiki URL: http://wiki.icub.org *Baxter* Wiki URL: http://api.rethinkrobotics.com

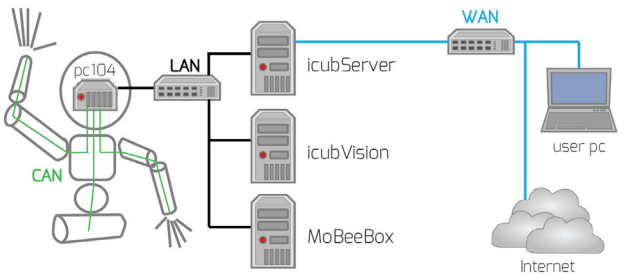


**FIGURE 3 | A sketch of the computing setup we used to operate the *iCub* at the IDSIA Robotics Lab**. The `pc104` handles the on-board data processing and controls the motors via CAN-bus. The `icubServer` is running the YARP server and is the router into the IDSIA-wide network and the internet. Dedicated computers for vision (`icubVision`) and collision avoidance (`MoBeeBox`) are used.

multiple computers. In our setup, the various computational tasks were implemented as nodes, which were then distributed throughout the network of computers. For the experiments on the *iCub*, a separate computer was used to run multiple object detection modules in parallel, while another computer (`MoBeEBox`) performed the collision avoidance and visualizing the world model. During the development of new modules, an additional user PC was connected via Ethernet to run and debug the new modules. Component-level abstraction using middleware increases portability across different robotic systems. For example, running *MoBeE* with different robot arms is easily done by simply providing the new arm's kinematic model as an XML file. Transferring to other middleware systems is also possible, though a bit more intricate. We have ported various parts of the architecture to ROS-based modules allowing to interact with ROS-based robots, such as *Baxter*.

Humanoid robots, and the *iCub* in particular, have a high DOF, which allows for complex motions. To perform useful actions, many robots need to be controlled in unison requiring robust control and planning algorithms. Our framework consists of an action subsystem, which in turn contains collision avoidance and grasping capabilities.

## 2.1. Object Detection and Localization Modules: icVision

Our humanoid robot should be able to learn how to perceive and detect objects from very few examples, in a manner similar to humans. It should have the ability to develop a representation that allows it to detect the same object again and again, even when the lighting conditions change, e.g., during the course of a day. This is a necessary prerequisite to enable adaptive, autonomous behaviors based on visual feedback. Our goal is to apply a combination of robot learning approaches, artificial intelligence, and machine learning techniques, with computer vision, to enable a variety of proposed tasks for robots.

*icVision* (Leitner et al., 2013c) was developed to support current and future research in cognitive robotics. This follows a "passive" approach to the understanding of vision, where the actions of the human or robot are not taken into account. It processes the visual inputs received by the cameras and builds (internal) representations of objects. This computation is distributed over multiple modules. It facilitates the 3D localization of the detected objects in the 2D image plane and provides this information to other systems, e.g., a motion planner. It allows to create distributed systems of loosely coupled modules and provides standardized interfaces. Special focus is put on object detection in the received input images. **Figure 4** shows how a simple red detection can be added as a separate running module. Specialized modules, containing a specific model, are used to detect distinct patterns or objects. These specialized modules can be connected and form pathways to perform, e.g., object detection, similarly to the hierarchies in the visual cortex. While the focus herein is on the use of single and stereo camera images, we are confident that information from RGB-D cameras (such as the Microsoft Kinect) can be easily integrated.

The system consists of different modules, with the core module providing basic functionality and information flow. **Figure 5** shows separate modules for the detection and localization and their connection to the core, which abstract the robot's cameras and the communication to external agents. These external agents are further modules and can do a wide variety of tasks, for example, specifically test and compare different object detection or localization techniques. *icVision* provides a pipeline that connects visual perception with world modeling in the *MoBeE* module (dashed line in **Figure 5**). By processing the incoming images from the robot with a specific filter for each "eye," the location of the specific object can be estimated by the localization module and then communicated to *MoBeE* (**Figure 6** depicts the typical information flow).

## 2.2. Robot and World Modeling for Collision Avoidance: MoBeE

*MoBeE* (Modular Behavior Environment for Robots) is at the core of the described framework for eye–hand coordination. It is a solid, reusable, open-source[3] toolkit for prototyping behaviors on the *iCub* humanoid robot. *MoBeE* represents the state-of-the-art in humanoid robotic control and is similar in conception to the control system that runs DLR's Justin (De Santis et al., 2007; Dietrich et al., 2011). The goal of *MoBeE* is to facilitate the close integration of planning and motion control (sketched in **Figure 2B**). Inspired by Brooks (1991), it aims to embody the planner, provide safe and robust action primitives, and perform real-time re-planning. This facilitates exploratory behavior using a real robot with *MoBeE* acting as a supervisor preventing collisions, even between multiple robots. It consists of three main parts all implemented in C++: a kinematic library with a visualization, and a controller, running in two separate modules. These together provide the "collision avoidance" (yellow) and "world model" (blue) as depicted in **Figure 2A**. **Figure 5** shows the connections between the various software entities required to run the full eye–hand coordination framework. *MoBeE* communicates

[3]URL: https://github.com/kailfrank/MoBeE

```
#include <string>
#include <yarp/os/all.h>
#include <yarp/sig/all.h>

#include "icFilterModule.h"

using namespace std;
using namespace yarp::os;
using namespace yarp::sig;

class RedFilterModule : public icFilterModule {

public:
    RedFilterModule() { setName("RedFilterTest"); }
    ~RedFilterModule() {}

    /* This is our main function. Will be called periodically every getPeriod() s */
    bool TestModule::updateModule()
    {
        if(! isRunning ) return false;

        ImageOf<PixelBgr> *left_image = leftInPort.read();  // read an image
        ... // check for valid image

        // very simple test for red-ness
        for (int x=0; x < left_image->width(); x++)
            for (int y=0; y < left_image->height(); y++) {
                PixelBgr& pixel = left_image->pixel(x,y);
                // make sure red level exceeds blue and green by a factor and a threshold
                if ( pixel.r > pixel.b * 1.5 + 10 && pixel.r > pixel.g * 1.5 + 10 )
                    output_image[x][y] = 1;
                else
                    output_image[x][y] = 0;
            }

        outputPort.write();         // write the output image
        return isRunning;
    }
};
```

**FIGURE 4 | Little coding is required for a new module to be added as filter to *icVision*.** This shows a simple red filter being added. The image acquisition, connection of the communication ports, and cleanup are all handled by the superclass.

with the robot and provides an interface to other modules. One of these is the perception side *icVision*.

In its first iteration, *MoBeE* provided virtual feedback for a reinforcement learning experiment. This was necessary as most current robotic systems lack a physical skin that would provide sensory information to perform reflexive motions. It was intended to enforce constraints in real time while a robot is under the control of any arbitrary planner/controller. This led to a design based on switching control, which facilitated experimentation with pre-existing control modules. A kinematic model is loaded from an XML file using "Zero Position Displacement Notation" (Gupta, 1986).

When the stochastic or exploratory agent/controller (light gray at the top in **Figure 5**) does something dangerous or undesirable, *MoBeE* intervenes. Collision detection is performed on the loaded kinematic robot model consisting of a collection of simple geometries to form separate body parts (see **Figure 7**). These geometries are created as C++ objects that inherit functionality from both the fast geometric intersection library and the visualization in OpenGL. The joint encoders provided by the robot abstraction layer are used to calculate collisions, i.e., intersecting body parts. In the first version, this collision signal was used to avoid collisions by switching control, which was later abandoned in favor of a second-order dynamical system (Frank, 2014). Constraints, such as impeding collisions, joint limits, or cable lengths, can be addressed by adding additional forces to the system. Due to the dynamical system, many of the collisions encountered in practice no longer stop the robot's action, but rather deflect the requested motion, bending it around an obstacle.

*MoBeE* continuously mixes control and constraint forces to generate the robot motion in real time and results in smoother, more intuitive motions in response to constraints/collisions (**Figure 8**). The effects of sensory noise are mitigated passively by the controller. The constraint forces associated with collisions are proportional to their penetration depth; in the experimentation, it was observed that the noise in the motor encoder signal has a minimal effect on collision response. The sporadic shallow collisions, which can be observed when the robot is operating close to an obstacle, such as the other pieces of a chess board, generate tiny forces that only serve to nudge the robot gently away from the obstacle. *MoBeE* in addition can be used for adaptive roadmap planning (Kavraki et al., 1996; Stollenga et al., 2013), the dynamical approach means that the planner/controller is free to explore continuous spaces, without the need to divide them into safe and unsafe regions.

The interface for external agents is further simplified by allowing to subscribe to specific points of interest in the imported models (seen in yellow in **Figure 7**). These markers can be defined both on static or moving objects or the robots. The marker positions or events, such as the body part being in a colliding pose,

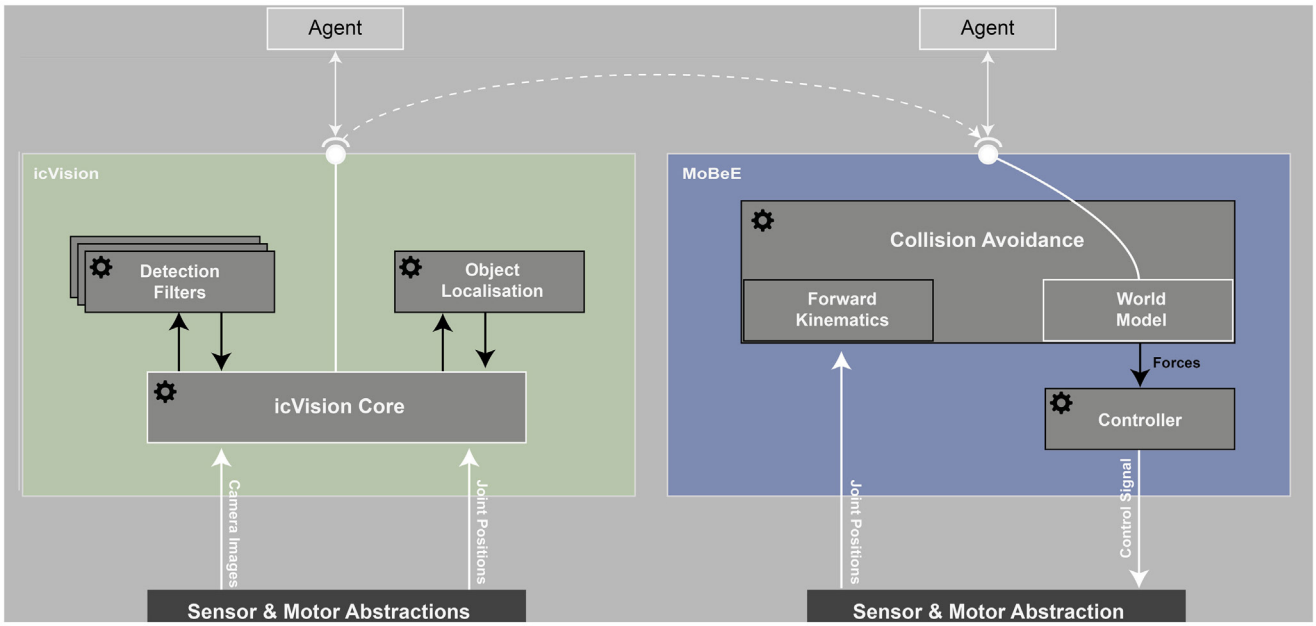


**FIGURE 5 | The framework described consists of multiple software entities (indicated by cogwheels), all connected via a communication layer, such as YARP or ROS**. The provided robot abstraction is used by the main modules. On the perception side, *icVision* processes the incoming camera images in its core module and sends them to (possibly multiple) separately running detection filters. Another separate entity is performing the localization based on the detected objects and robot's pose. The *icVision* Core also provides interfaces for agents to query for specific objects or agents that learn object representations (such as a CGP-IP based learner). The interface also provides the objects location to *MoBeE*. There a world model is created by calculating the forward kinematics from the incoming joint positions. The same entity performs the collision avoidance between separate body parts or the objects that have been detected by *icVision*. The controller is independent and translates the virtual forces created by *MoBeE* or provided by higher level planning agents into motor commands.

are broadcast via the interface allowing connected agents to react, e.g., to trigger a grasp primitive. More details about the whole *MoBeE* architecture and how it was used for reach learning can be found in Frank (2014). Additionally, we have published multiple videos of our robotic experiments while using *MoBeE* foremost: "Toward Intelligent Humanoids."[4]

## 2.3. Action Repertoire: LEOGrasper Module

Robotic grasping is an active and large research area in robotics. A main issue is that in order to grasp successfully the pose of the object to be grasped has to be known quite precisely. This is due to the grasp planners required to plan the precise placement and motion of each individual "finger" (or gripper). Several methods for robust grasp planning exploit the object geometry or tactile sensor feedback. However, object pose range estimation introduces specific uncertainties that can also be exploited to choose more robust grasps (Carbone, 2013).

A different approach is used in our implementation that does use a more reactive approach. Grasp primitives are triggered from *MoBeE*, which involve the controlling the five digit *iCub* hand. These primitives consist of target points in joint space to be reached sequentially during grasp execution. Another problem is to realize when to stop grasping. The *iCub* has touch sensors on the palm and finger tips. To know when there is a successful grasp, these sensors need to be calibrated for the material in use. Especially for objects as varied as plastic cups, ceramic mugs, and tin cans, the tuning can be quite cumbersome and leads to a lower signal-to-noise ratio. We decided to overcome this by using the errors from the joint controllers directly. This approach allows to provide feedback whether a grasp was successful or not to a planner or learning system.

*LEOGrasper* is our light-weight, easy-to-use, one-shot grasping system for the *iCub*. The system itself is contained in one single module using YARP to communicate. It can be triggered by a simple command from the command line, network, or as in our case from *MoBeE*. The module can be configured for multiple grasp types, these are loaded from a simple text file, containing some global parameters (such as the maximum velocity) as well as the trajectories. Trajectories are specified by providing positions for each joint individually, containing multiple joints per digit as well as abduction, spread, etc. on the *iCub*. We provide power and pinch grasp and pointing gestures. For example, to close all digits in a coordinated fashion, at least two positions need to be defined, the starting and end position (see **Figure 9**). For more intricate grasps, multiple intermediate points can be provided. The robot's fingers are controlled from the start point to each consecutive point, when an `open` signal is received. For `close`, the points are sent in reverse order.

[4]Webpage: http://Juxi.net/media/ or direct video URL: http://vimeo.com/51011081

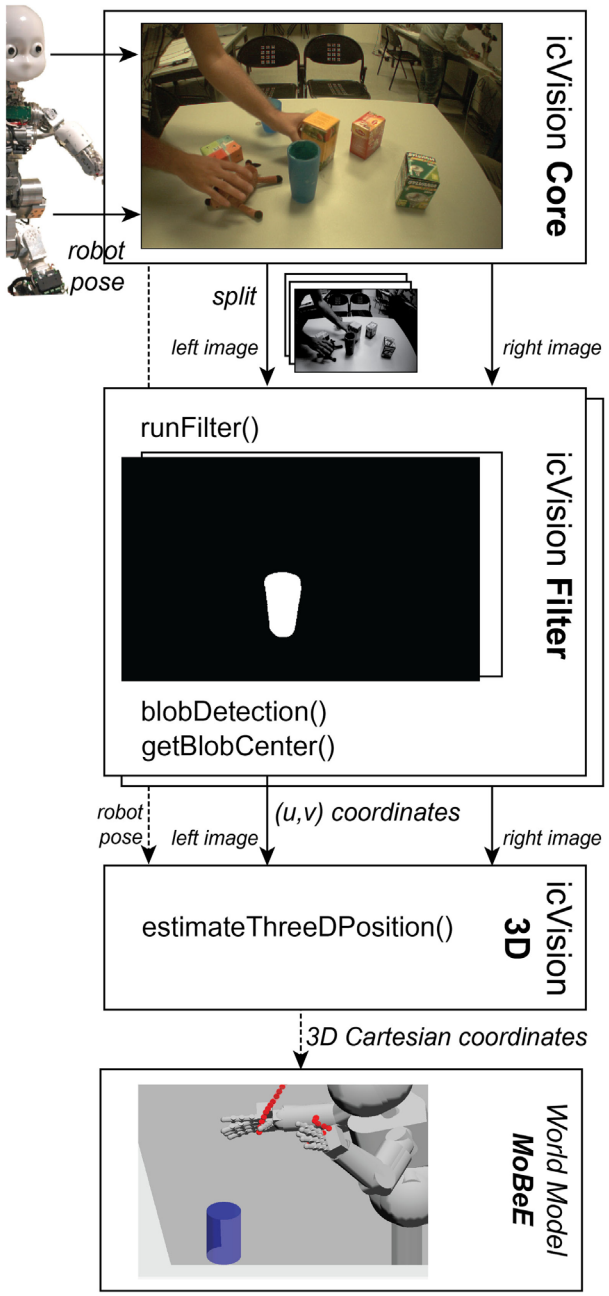


**FIGURE 6 | To provide information about the 3D location of an object to *MoBeE*, the following is performed: at first, camera images are received by the *core* from the hardware via a communications layers.** The images are split into channels and made available to each individual *icVision* filters that is currently active. These then perform binary segmentation for a specific object. The objects (center) location in the image frame, *(u,v)* is then communicated to a 3D localization module. Using the joint encoder values and the object's location in both eyes, a location estimate is then sent to the *MoBeE* world model (Leitner et al., 2013a).

*LEOGrasper* has been used extensively in our robotics lab and selected successful grasps are shown in **Figure 10**.[5] The existing trajectories and holding parameters were tuned through experimentation; in the future, we aim at learning these primitives using human demonstrations or reward signals.

## 3. METHOD OF INTEGRATING ACTION AND VISION: APPLYING THE FRAMEWORK

The framework has been extensively used over the last few years in our experimental robotics research. Various papers have been published during the development of the different subsystems and their improvements. **Table 1** provides an overview. *MoBeE* can be pre-loaded with a robot model using an XML file that describes the kinematics based on "Zero Position Displacement Notation" (Gupta, 1986). **Figure 11** shows a snippet from the XML describing the Katana robotic arm. In addition a pre-defined, marked-up world model can be loaded from a separate file as well. This is particularly useful for stationary objects in the world or to restrict the movement space of the robot during learning operations.

Through the common interface to *MoBeE* object properties of each object can be modified, through an RPC call, following YARP standard and is accessible from the command line, a webpage, or any other module connecting to it. These objects are placed in the world model by either loading from a file at start-up or during runtime by agents, such as the *icVision* core. Through the interface an object can also be set as an *obstacle*, which means repelling forces are calculated, or as a *target*, which will attract the end-effector. In addition, objects can be defined as ghosts, leading to the object being ignored in the force calculation.

As mentioned earlier on, previous research suggests that connections between motor actions and observations exist in the human brain and describes their importance to human development (Berthier et al., 1996). To interface and connect artificial systems performing visual and motor cortex-like operations on robots will be crucial for the development of autonomous robotic systems. When attempting to learn behaviors on a complex robot, such as the *iCub* or Baxter, state-of-the-art AI and control theories can be tested (Frank et al., 2014) and shortcomings of these learning methods can be discovered (Zhang et al., 2015) and addressed. For example, Hart et al. (2006) showed that a developmental approach can be used for a robot to learn to reach and grasp. We developed modules for action generation and collision avoidance and their interfaces to the perception side. By having the action and motion side tightly coupled, we can use learning algorithms that require also negative feedback. We can create this without actually "hurting" the robot.

### 3.1. Example: Evolving Object Detectors

We previously developed a technique based on Cartesian Genetic Programing (CGP) (Miller, 1999, 2011) allowing for the automatic generation of computer programs for robot vision tasks, called Cartesian Genetic Programming for Image Processing (CGP-IP) (Harding et al., 2013). CGP-IP draws inspiration from previous work in the field of machine learning and combines it with the available tools in the image processing discipline, namely in the form of OpenCV functions. OpenCV is an open-source

[5]Source code available at: https://github.com/Juxi/iCub/

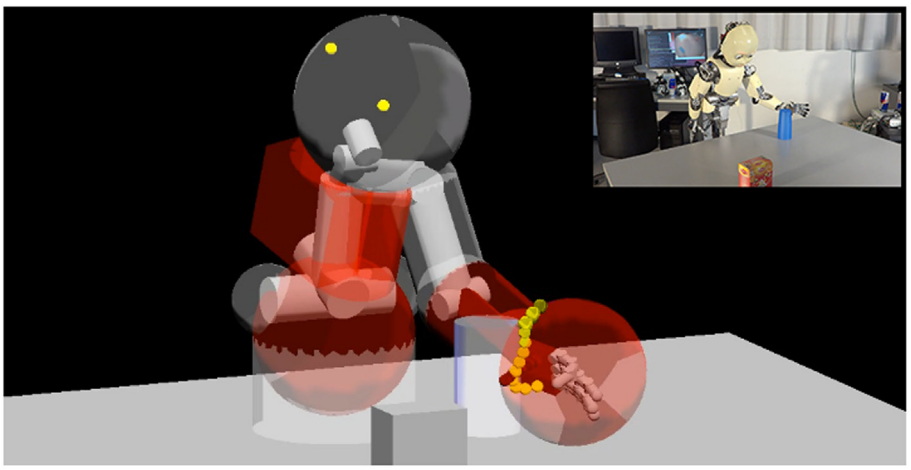

**FIGURE 7 | A scene of the *iCub* avoiding an object (inset) during one of our experiments (Leitner et al., 2014b) and its corresponding visualization of the *MoBeE* model**. Red body parts are highlighting impeding collisions with either another body part (as in the case of the hip with the upper body) or an object in the world model (hand with the cup). (See video: https://www.youtube.com/watch?v=w_qDH5tSe7g).

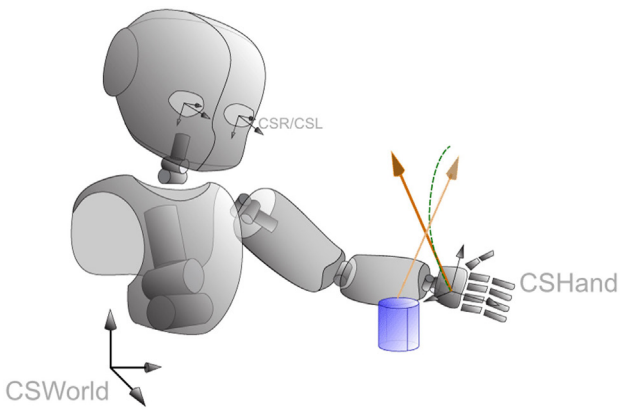


**FIGURE 8 | The virtual forces created by the dynamical system within *MoBeE***. It continuously mixing control and constraint forces (orange vectors) to generate the robot motion in real time. It results in a smoother, more intuitive motions in response to constraints/collisions (dashed green line). To calculate the force, the distance of the object to the hand in its coordinate frame CSHand is used. *MoBeE* handles the transformation from the coordinate systems of the cameras (CSR/CSL) to the world frame CSWorld and CSHand.

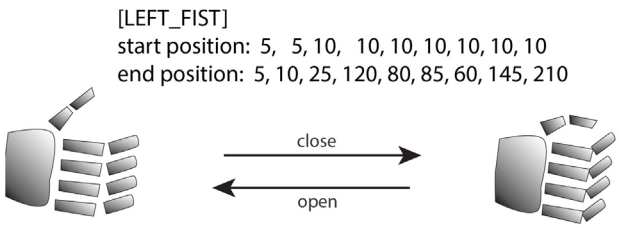


**FIGURE 9 | Defining a grasp in *LEOGrasper* is simple: defining a start and end position in joint space is all that is required**. The `open` command will revert the hand into the start state (left), `close` will attempt to reach the end state (right). For more complex grasps intermediate states can be provided.

framework providing a mature toolbox for a variety of image processing tasks. This domain knowledge is integrated into CGP-IP allowing to quickly evolve object detectors from only a small training set – our experiments showed that just a handful of (5–20) images per object are required. These detectors can then be used to perform the binary image segmentation within the *icVision* framework. In addition, CGP-IP allows for the segmentation of color images with multiple channels, a key difference to much of the previous work focusing on gray scale images. CGP-IP deals with separate channels and splits incoming color images into individual channels before they can be used at each node in the detector. This leads to the evolutionary process selecting which channels will be used and how they are combined.

CGP-IP manages a population of candidates, which consists of individual *genes*, representing the nodes. Single channels are used as inputs and outputs of each node, while the action of each node is the execution of an OpenCV function. The full candidate can be interpreted as a computer program performing a sequence of image operations on the input image. The output of each candidate filter is a binary segmentation. GPs are supervised, in the sense that a fitness will need to be calculated for each candidate. For scoring each individual, a ground truth segmentation needs to be provided. A new generation of candidates is then created out of the fittest individuals. An illustrative example of a CGP-IP candidate is shown in **Figure 12**. CGP-IP can directly create C# or C++ code from these graphs. The code can be executed directly on the real hardware or pushed as updates to existing filter modules running within *icVision*. CGP-IP includes classes for image operations, the evolutionary search and the integration


**FIGURE 10 | The *iCub* hand during grasp execution with a variety of objects, including tin cans, tea boxes, and plastic cups (Leitner et al., 2014b).**

**TABLE 1 | Overview of experiments facilitated by parts of the architecture presented herein.**

| Experiment description | Framework | Reference |
|---|---|---|
| Autonomous object detection | icVision, CGP-IP | Leitner et al. (2012a, 2013b) |
| Multi robot collision avoidance | MoBeE (vSkin) | Leitner et al. (2012b,c) |
| Safe policy learning | MoBeE (vSkin) | Pathak et al. (2013) |
| Object detection and localisation | icVision | Leitner et al. (2013a) |
| Spatial perception learning | MoBeE, icVision | Leitner et al. (2013d) |
| Learning object detection | CGP-IP | Leitner et al. (2013e) |
| Humanoid motion planning | MoBeE | Stollenga et al. (2013) |
| Reinforcement learning for reaching | MoBeE | Frank et al. (2014) |
| Improving vision through interaction | Full system | Leitner et al. (2014a) |
| Reactive reaching and grasping | Full system | Leitner et al. (2014b) |
| Cognitive and developmental robots | Full system | Leitner (2015) |

with the robotic side through a middleware. Currently, we are extending the C# implementation to run on various operating systems and be integrated into a distributed visual system, such as DRVS (Chamberlain et al., 2016).

CGP-IP allows not just for a simple reusable object detection but also provides a simple way of learning these based on only very small training sets. In connection with our framework, these data can be collected on the real hardware and the learned results directly executed. For this, an agent module was designed that communicates with the *icVision* core through its interface. Arbitrary objects are then placed in front of the robot and images are collected while the robot is moving about. The collected images are then processed by the agent and a training set is created. With the ground truth of the location known, so is the location of the object in the image. A fixed size bounding box around this location leads to the ground truth required to evolve an object detector. This way the robot (and some prior knowledge of the location) can be used to autonomously learn object detectors for all the objects in the robot's environment (Leitner et al., 2012a).

## 3.2. Example: Reaching While Avoiding a Moving Obstacle

The inverse kinematics problem, i.e., placing the hand at a given coordinate in operational space, can be performed with previously available software on the *iCub*, such as the existing operational space controller (Pattacini, 2011) or a roadmap-based approach (Stollenga et al., 2013). These systems require very accurate knowledge of the mechanical system to lead to precise solutions, requiring a lengthy calibration procedure. These systems also tend to be brittle when change in the robot's environment requires adapting the created motions.

To overcome this problem, the framework, as described above, creates virtual forces based on the world model within *MoBeE* to govern the actual movement of the robot. Static objects in the environment, such as, e.g., the table in front of the robot, can be added directly into the model via an XML file. Once in the model, actions and behaviors are adapted due to computed constraint forces. This way we are able to send arbitrary motions to our system, while ensuring the safety of our robot. Even with just these static objects, this has been shown to provide an interesting way to learn robot reaching behaviors through reinforcement (Pathak et al., 2013; Frank et al., 2014). The presented system has the same functionality also for arbitrary, non-static objects.

For this after the detection in both cameras, the object's location is estimated and updated in the world model. The forces are continually recalculated to avoid impeding collisions even with moving objects. **Figure 7** shows how the localized object is in the way of the arm and the hand. To ensure the safety of the rather fragile fingers, a collision sphere around the end-effector was added – seen in red, indicating a possible collision due to the sphere intersecting with the object. The same can be seen with the lower arm. The forces push the intersecting geometries away from each other, leading to a movement of the

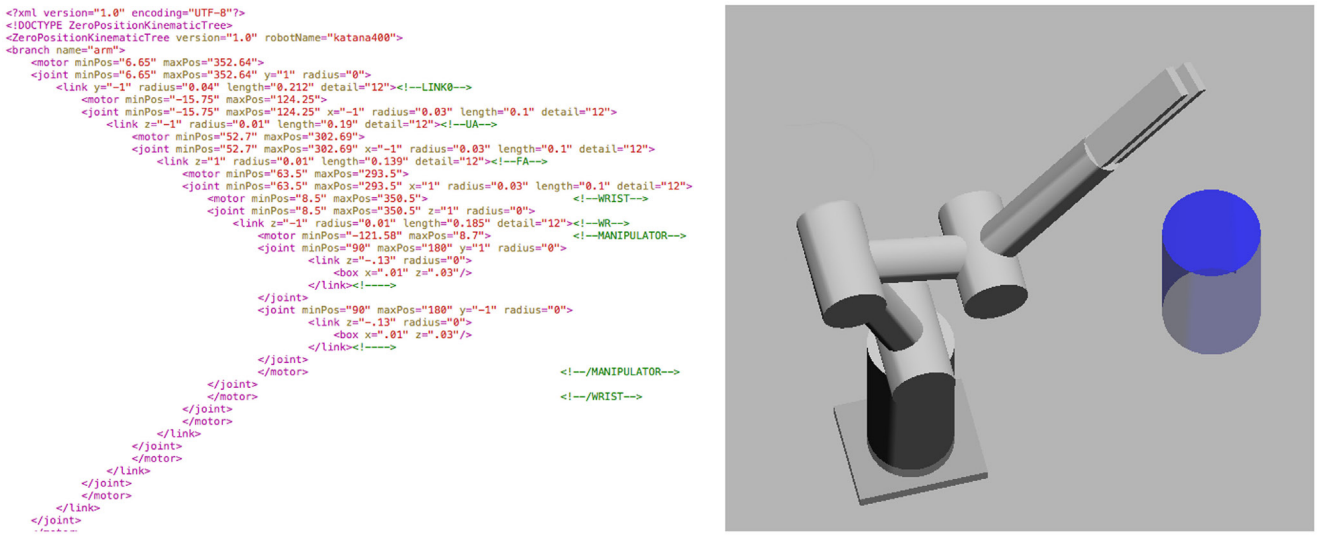


**FIGURE 11 | (Left) The XML files used to describe the kinematics of the Katana arm**. (Right) Visualization of the same XML file in *MoBeE*.

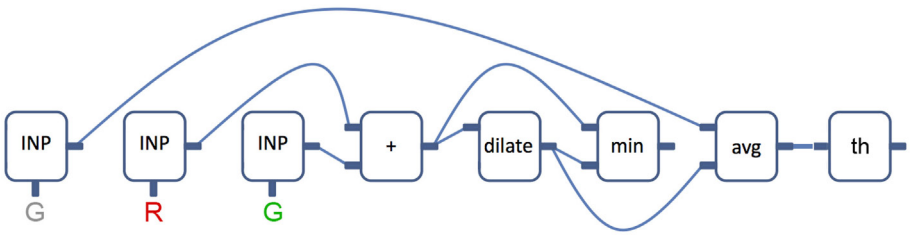


**FIGURE 12 | Example illustration of a CGP-IP genotype, taking three input channels of the current image**. A sequence of OpenCV operations is then performed before thresholding occurs to produce a binary segmented output image.

end-effector away from the obstacle. **Figure 13** shows how the robot's arm is "pushed" aside when the cup is moved close to the arm, therefore avoiding a non-stationary obstacle. It does so until the arm reaches its limit, then the forces cumulate and the end-effector is "forced" upwards to continue avoiding the obstacle. Similarly the reaching behaviour is adapted while the object is moved. Without an obstacle, the arm starts to settle back into its resting pose $q^*$. By simply sending a signal through the interface the type of the object within the world model can be changed from `obstacle` into `target`. This leads to the calculated forces now being attracting not repelling. *MoBeE* also allows to trigger certain responses when collisions occur. In the case when we want the robot to pick-up the object, we can activate a grasp subsystem whenever the hand is in the close vicinity of the object. We are using a prototypical power grasp style hand-closing action, which has been used successfully in various demos and videos.[6] **Figure 10** shows the *iCub* successfully picking up (by adding an extra upwards force) various objects using our grasping subsystem, executing the same action. Our robot frameworks are able to track multiple objects at the same time, which is also visible in **Figure 7**, where both the cup and the tea box are tracked. By simply changing the type of the object within *MoBeE*, the robot reaches for a certain object while avoiding the other.

## 3.3. Example: Improving Robot Vision by Interaction

The two subsystems can further be integrated for the use of higher level agents controlling the robot's behavior. Based on the previous section, the following example shows how an agent can be used to learn visual representations (in CGP-IP) by having a robot interact with its environment. Building on the previously mentioned evolved object detectors, we extended the robot's interaction ability to become better at segmenting objects. Similar to the experiment by Welke et al. (2010), the robot was able to curiously rotate the object of interest with its hand. Additional actions were added for the robot to perform, such as poke, push, and a simple viewpoint change by leaning left and right. Furthermore, a baseline image dataset is collected, while the robot (and the object) is static. In this experiment, we wanted to measure the impact of specific actions on the segmentation

[6]See videos at: http://Juxi.net/media/

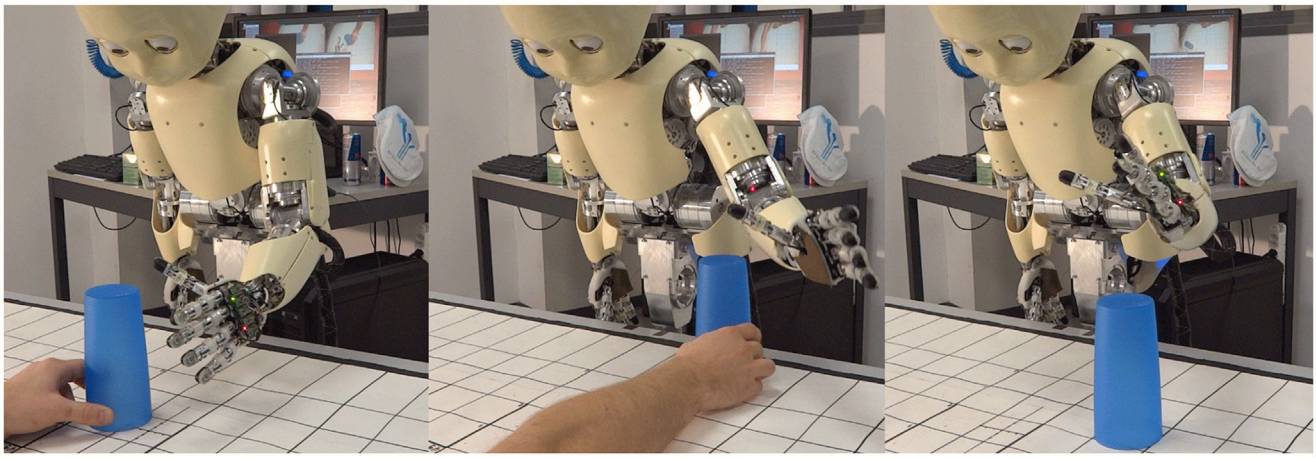

**FIGURE 13 | The *iCub*'s arm is controlled by *MoBeE* to stay in a non-colliding pose of the moving obstacle and the table by using reactive virtual forces**. (See video: https://www.youtube.com/watch?v=w_qDH5tSe7g).

performance. After the robot performed one of the four pre-programed actions, a new training set was collected, which contains the images from the static scenario and the images during action execution. While more data mean generally better results, we could also see that some actions were leading to better results than others. **Figure 14** shows visually how the improvement leads to better object segmentation, in validation images. On the left is the original camera image, in the middle the segmentation performed by an evolved filter solely based on the "static scene baseline," and on the right is the segmentation when integrating the new observations during a haptic exploration action.

By providing a measurable improvement, the robot can select and perform the action yielding the best possible improvement for a specific detector. Interaction provides robots with a unique possibility (compared to cameras) to build more accurate and robust visual representations. Simple leaning actions change the camera viewpoint sufficiently to collect a different dataset. This does not just help with separating geometries of the scene but also creates more robust and discriminative classifiers. Active scene interaction by e.g., applying forces to objects enables the robot to start to reason about relationships between objects, such as "are two objects (inseparably) connected," or, find out other physical properties, like, "is the juice box full or empty," We are planning to add more complex actions and abilities to learn more object properties and have started to investigate how to determine an object's mechanical properties through interaction and observation (Dansereau et al., 2016).

## 4. DISCUSSION

Herein, we present our modular software framework applied in our research toward autonomous and adaptive robotic manipulation with humanoids. A tightly integrated sensorimotor system, based on subsystems developed over the past years, enables a basic level of eye–hand coordination on our robots. The robot detects objects, placed at random positions on a table, and performs a visually guided reaching before executing a simple grasp.

Our implementation enables the robot to adapt to changes in the environment. It safeguards complex humanoid robots, such as the *iCub*, from unwanted interactions – i.e., collisions with the environment or itself. This is performed by integrating the visual system with the motor side by applying attractor dynamics based on the robot's pose and a model of the world. We achieve a level of integration between visual perception and actions not previously seen on the *iCub*. Our approach, while comparable to visual servoing, has the advantage of being completely modular and the ability to take collisions (and other constraints) into account.

The framework has grown over recent time and has been used in a variety of experiments mainly with the *iCub* humanoid robot. It has since then been ported in parts to work with ROS with the aim of running pick and place experiments on Baxter; the code will be made available on the authors webpage at: http://juxi.net/projects/VisionAndActions/. The overarching goal was to enable a way of controlling a complex humanoid robot, which combines motion planning with low-level reflexes from visual feedback. *icVision* provides the detection and localization of objects in the visual stream. For example, it will provide the location of a chess board on a table in front of the robot. It can also provide the position of chess pieces to the world model. Based on this, an agent can plan a motion to pick up a specific piece. During the execution of that motion, *MoBeE* calculates forces for each chess piece, attracting for the target piece, repelling forces for all the other pieces. These forces are updated whenever a new object (or object location) is perceived, yielding a more robust execution of the motion due to a better coordination between vision and action.

The current system consists of a mix of pre-defined and learned parts, in the future, we plan to integrate further machine learning techniques to improve the object manipulation skills of robotic systems. For example, learning to plan around obstacles, including improved prediction and selection of actions. This will lead to a more adaptive, versatile robot, being able to work in unstructured, cluttered environments. Furthermore, it might be of interest to investigate an even tighter sensorimotor coupling,

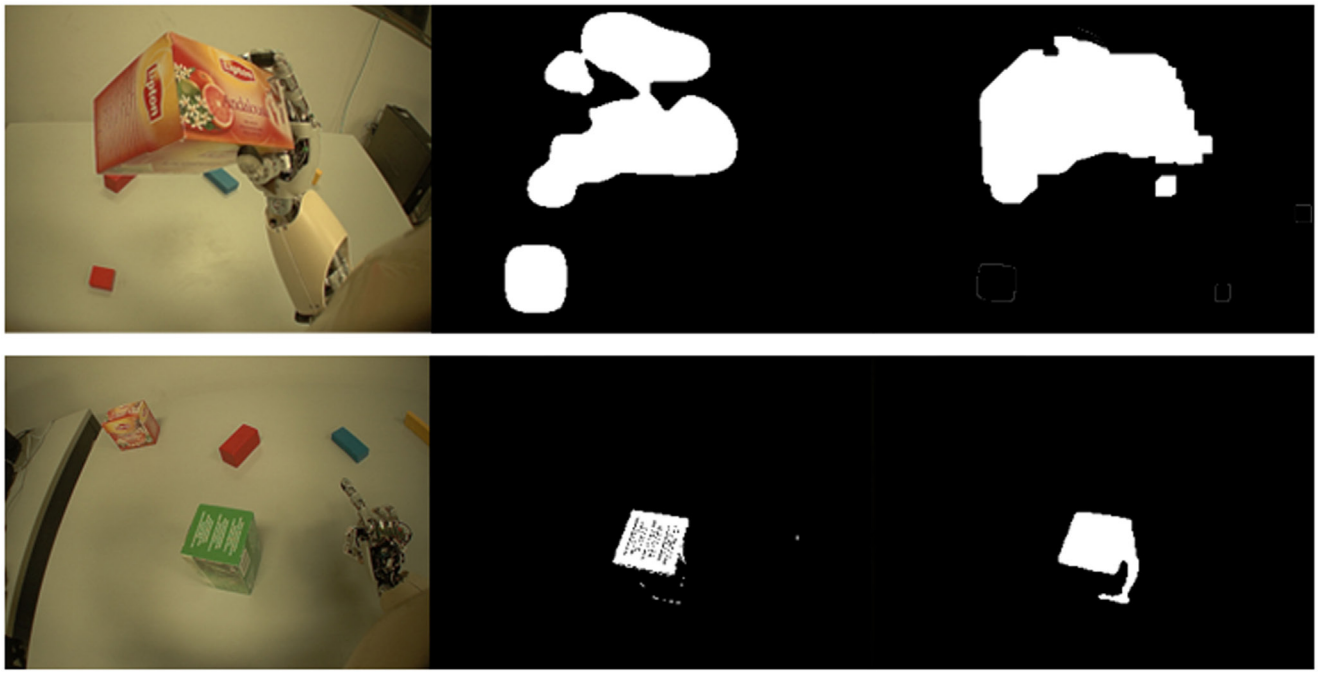

**FIGURE 14 | Segmentation improvements for two objects after interaction**. On the left, the robot's view of the scene. The middle column shows the first segmentation generated from the "static scene baseline." The last column shows the improved segmentation after learning continued with new images collected during the manipulation of the object.

e.g., by working directly in the image space – similar to image-based visual servoing approaches (Chaumette and Hutchinson, 2006) – this way avoiding to translate 2D image features into operational space locations.

In the future, we are aiming to extend the capabilities to allow for the quick end-to-end training of reaching (Zhang et al., 2015) and manipulation tasks (Levine et al., 2015), as well as, easy transition from simulation to real-world experiments. We are also looking at developing agents that interface this framework to learn the robot's kinematics and adapt to changes occurring due to malfunction or wear, leading to self calibration of a robot's eye–hand coordination.

## AUTHOR CONTRIBUTIONS

JL is the main contributor both in the research and software integration of the framework. SH designed the CGP-IP software framework and the related experiments. AF designed and contributed to the research experiments on the iCub. PC contributed to the manuscript and the research experiments on Baxter.

## ACKNOWLEDGMENTS

The authors would like to thank Mikhail Frank, Marijn Stollenga, Leo Pape, Adam Tow, and William Chamberlain for their discussions and valued inputs to this paper and the underlying software frameworks presented herein.

## FUNDING

Various European research projects (IM-CLeVeR #FP7-IST-IP-231722, STIFF #FP7-IST-IP-231576) and the Australian Research Council Centre of Excellence for Robotic Vision (#CE140100016).

## REFERENCES

Ambrose, R., Wilcox, B., Reed, B., Matthies, L., Lavery, D., and Korsmeyer, D. (2012). *NASA's Space Technology Roadmaps (STRs): Robotics, Tele-Robotics, and Autonomous Systems Roadmap*. Technical Report. Washington, DC: National Aeronautics and Space Administration (NASA).

Ando, N., Suehiro, T., and Kotoku, T. (2008). "A software platform for component based RT-system development: Openrtm-aist," in *Simulation, Modeling, and Programming for Autonomous Robots*, eds S. Carpin, I. Noda, E. Pagello, M. Reggiani and O. von Stryk (Springer), 87–98.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). "SURF: speeded up robust features," in *Computer Vision ECCV 2006*, eds A.Leonardis, H.Bischof, and A.Pinz (Berlin Heidelberg: Springer), 404–417.

Berthier, N., Clifton, R., Gullapalli, V., McCall, D., and Robin, D. (1996). Visual information and object size in the control of reaching. *J. Mot. Behav.* 28, 187–197. doi:10.1080/00222895.1996.9941744

Brooks, R. (1991). Intelligence without representation. *Artif. Intell.* 47, 139–159. doi:10.1016/0004-3702(91)90053-M

Brown, E., Rodenberg, N., Amend, J., Mozeika, A., Steltz, E., Zakin, M., et al. (2010). Universal robotic gripper based on the jamming of granular

material. *Proc. Natl. Acad. Sci. U.S.A.* 107, 18809–18814. doi:10.1073/pnas.1003250107

Brugali, D. (2007). *Software Engineering for Experimental Robotics*, Vol. 30. Berlin; Heidelberg: Springer.

Carbone, G. (2013). *Grasping in Robotics, Volume 10 of Mechanisms and Machine Science*. London: Springer.

Chamberlain, W., Leitner, J., and Corke, P. (2016). "A distributed robotic vision service," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Stockholm.

Chaumette, F., and Hutchinson, S. (2006). Visual servo control, part I: basic approaches. *IEEE Robot. Autom. Mag.* 13, 82–90. doi:10.1109/MRA.2006.250573

Ciliberto, C., Smeraldi, F., Natale, L., and Metta, G. (2011). "Online multiple instance learning applied to hand detection in a humanoid robot," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. San Francisco, CA.

Cipolla, R., Battiato, S., and Farinella, G. M. (2010). *Computer Vision: Detection, Recognition and Reconstruction*, Vol. 285. Berlin Heidelberg: Springer.

Corke, P. (2011). *Robotics, Vision and Control, Volume 73 of Springer Tracts in Advanced Robotics*. Berlin; Heidelberg: Springer.

Dansereau, D. G., Singh, S. P. N., and Leitner, J. (2016). *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Stockholm.

Davison, A. J., and Murray, D. W. (2002). Simultaneous localization and map-building using active vision. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 865–880. doi:10.1109/TPAMI.2002.1017615

De Santis, A., Albu-Schäffer, A., Ott, C., Siciliano, B., and Hirzinger, G. (2007). "The skeleton algorithm for self-collision avoidance of a humanoid manipulator," in *Advanced Intelligent Mechatronics, 2007 IEEE/ASME International Conference on* (Zürich: IEEE), 1–6.

Diankov, R., and Kuffner, J. (2008). *Openrave: A Planning Architecture for Autonomous Robotics*. Tech. Rep. CMU-RI-TR-08-34. Pittsburgh, PA: Robotics Institute, 79.

Dietrich, A., Wimbock, T., Taubig, H., Albu-Schaffer, A., and Hirzinger, G. (2011). "Extensions to reactive self-collision avoidance for torque and position controlled humanoids," in *Proceedings of the International Conference on Robotics and Automation (ICRA)* (Shanghai: IEEE), 3455–3462.

Elkady, A., and Sobh, T. (2012). Robotics middleware: a comprehensive literature survey and attribute-based bibliography. *J. Robot.* 2012. doi:10.1155/2012/959013

Fanello, S. R., Ciliberto, C., Natale, L., and Metta, G. (2013). "Weakly supervised strategies for natural object recognition in robotics," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*. Karlsruhe.

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Rob. Auton. Syst.* 56, 29–45. doi:10.1016/j.robot.2007.09.014

Forssberg, H., Eliasson, A., Kinoshita, H., Johansson, R., and Westling, G. (1991). Development of human precision grip I: basic coordination of force. *Exp. Brain Res.* 85, 451–457. doi:10.1007/BF00229422

Frank, M. (2014). *Learning to Reach and Reaching to Learn: A Unified Approach to Path Planning and Reactive Control through Reinforcement Learning*. Ph.D. thesis, Universitá della Svizzera Italiana, Lugano.

Frank, M., Leitner, J., Stollenga, M., Förster, A., and Schmidhuber, J. (2014). Curiosity driven reinforcement learning for motion planning on humanoids. *Front. Neurorobot.* 7:25. doi:10.3389/fnbot.2013.00025

Gerkey, B., Vaughan, R. T., and Howard, A. (2003). "The player/stage project: tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics, Volume 1* (Coimbra), 317–323.

Gori, I., Fanello, S., Odone, F., and Metta, G. (2013). "A compositional approach for 3D arm-hand action recognition," in *Proceedings of the IEEE Workshop on Robot Vision (WoRV)*. Clearwater, FL.

Gupta, K. (1986). Kinematic analysis of manipulators using the zero reference position description. *Int. J. Robot. Res.* 5, 5. doi:10.1177/027836498600500202

Harding, S., Leitner, J., and Schmidhuber, J. (2013). "Cartesian genetic programming for image processing," in *Genetic Programming Theory and Practice X, Genetic and Evolutionary Computation*, eds R.Riolo, E.Vladislavleva, M. D.Ritchie, and J. H.Moore (New York; Ann Arbor: Springer), 31–44.

Hart, S., Ou, S., Sweeney, J., and Grupen, R. (2006). "A framework for learning declarative structure," in *Proceedings of the RSS Workshop: Manipulation in Human Environments*. Philadelphia, PA.

Hartley, R., and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*, 2nd Edn. Cambridge, UK: Cambridge University Press.

Hutchinson, S., Hager, G. D., and Corke, P. I. (1996). A tutorial on visual servo control. *IEEE Trans. Robot. Automat.* 12, 651–670. doi:10.1109/70.538972

Jackson, J. (2007). Microsoft robotics studio: a technical introduction. *IEEE Robot. Autom. Mag.* 14, 82–87. doi:10.1109/M-RA.2007.905745

Jeannerod, M. (1997). *The Cognitive Neuroscience of Action*. Blackwell Publishing. Available at: http://au.wiley.com/WileyCDA/WileyTitle/productCd-0631196048.html

Johnson, M. H., and Munakata, Y. (2005). Processes of change in brain and cognitive development. *Trends Cogn. Sci.* 9, 152–158. doi:10.1016/j.tics.2005.01.009

Karlsson, N., Di Bernardo, E., Ostrowski, J., Goncalves, L., Pirjanian, P., and Munich, M. (2005). "The vSLAM algorithm for robust localization and mapping," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*. Barcelona.

Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Robot. Autom. Mag.* 12, 566–580. doi:10.1109/70.508439

Kemp, C., Edsinger, A., and Torres-Jara, E. (2007). Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robot. Autom. Mag.* 14, 20–29. doi:10.1109/MRA.2007.339604

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.* 5, 90. doi:10.1177/027836498600500106

Kragic, D., and Vincze, M. (2009). Vision for robotics. *Found. Trends Robot.* 1, 1–78. doi:10.1561/2300000001

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi:10.1038/nature14539

Leitner, J. (2014). *Towards Adaptive and Autonomous Humanoid Robots: From Vision to Actions*. Ph.D. thesis, Universitá della Svizzera italiana, Lugano.

Leitner, J. (2015). "Chapter 10, a bottom-up integration of vision and actions to create cognitive humanoids," in *Cognitive Robotics* (CRC Press), 191–214. Available at: https://www.crcpress.com/Cognitive-Robotics/Samani/9781482244564

Leitner, J., Chandrashekhariah, P., Harding, S., Frank, M., Spina, G., Förster, A., et al. (2012a). "Autonomous learning of robust visual object detection and identification on a humanoid," in *Proceedings of the International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. San Diego, CA

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2012b). Learning spatial object localization from vision on a humanoid robot. *Int. J. Adv. Robot. Syst.* 9.

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2012c). "Transferring spatial perception between robots operating in a shared workspace," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. Villamoura.

Leitner, J., Förster, A., and Schmidhuber, J. (2014a). "Improving robot vision models for object detection through interaction," in *International Joint Conference on Neural Networks (IJCNN)*. Beijing.

Leitner, J., Frank, M., Förster, A., and Schmidhuber, J. (2014b). "Reactive reaching and grasping on a humanoid: towards closing the action-perception loop on the icub," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (Vienna), 102–109.

Leitner, J., Harding, S., Chandrashekhariah, P., Frank, M., Förster, A., Triesch, J., et al. (2013a). "Learning visual object detection and localization using icVision," in *Biologically Inspired Cognitive Architectures 2012. Extended versions of selected papers from the Third Annual Meeting of the BICA Society (BICA 2012)*, eds. A. Chella, R. Pirrone, R. Sorbello and R. K. Jóhannsdóttir (Berlin Heidelberg: Springer), 29–41.

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2013b). "ALife in humanoids: developing a framework to employ artificial life techniques for high-level perception and cognition tasks on humanoid robots," in *Workshop on 'Artificial Life Based Models of Higher Cognition' at the European Conference on Artificial Life (ECAL)*. Taormina

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2013c). "An integrated, modular framework for computer vision and cognitive robotics research (icVision)," in *Biologically Inspired Cognitive Architectures 2012, Volume 196 of Advances in Intelligent Systems and Computing*, eds A. Chella, R. Pirrone, R. Sorbello, and K. Jóhannsdóttir (Berlin; Heidelberg: Springer), 205–210.

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2013d). "Artificial neural networks for spatial perception: towards visual object localisation in humanoid robots," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX: IEEE), 1–7.

Leitner, J., Harding, S., Frank, M., Förster, A., and Schmidhuber, J. (2013e). "Humanoid learns to detect its own hands," in *Proceedings of the IEEE Conference on Evolutionary Computation (CEC)* (Cancun), 1411–1418.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-End training of deep visuomotor policies. *J. Mac. Learn. Res.* 17, 1–40.

Lowe, D. (1999). "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision (ICCV)* Kerkyra.

Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., and Abbeel, P. (2010). "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *Proceedings of the International Conference on Robotics and Automation (ICRA)* (Anchorage, AK), 2308–2315.

Marchand, E., Spindler, F., and Chaumette, F. (2005). Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robot. Autom. Mag.* 12, 40–52. doi:10.1109/MRA.2005.1577023

McCarty, M., Clifton, R., Ashmead, D., Lee, P., and Goubet, N. (2001). How infants use vision for grasping objects. *Child Dev.* 72, 973–987. doi:10.1111/1467-8624.00329

Meltzoff, A. (1988). Infant imitation after a 1-week delay: long-term memory for novel acts and multiple stimuli. *Dev. Psychol.* 24, 470. doi:10.1037/0012-1649.24.4.470

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: yet another robot platform. *Int. J. Adv. Robot. Syst.* 3, 43–48. doi:10.5772/5761

Miller, J. (1999). "An empirical study of the efficiency of learning Boolean functions using a Cartesian genetic programming approach," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (Orlando, FL), 1135–1142.

Miller, J. F. (ed.). (2011). *Cartesian Genetic Programming: Natural Computing Series*. Berlin; Heidelberg: Springer.

Oztop, E., Bradley, N., and Arbib, M. (2004). Infant grasp learning: a computational model. *Exp. Brain Res.* 158, 480–503. doi:10.1007/s00221-004-1914-1

Pathak, S., Pulina, L., Metta, G., and Tacchella, A. (2013). "Ensuring safety of policies learned by reinforcement: reaching objects in the presence of obstacles with the icub," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. Tokyo.

Pattacini, U. (2011). *Modular Cartesian Controllers for Humanoid Robots: Design and Implementation on the iCub*. Ph.D. thesis, Italian Institute of Technology, Genova.

Plumert, J., and Spencer, J. (2007). *The Emerging Spatial Mind*. Oxford: Oxford University Press.

Posner, M. (1989). *Foundations of Cognitive Science*. Cambridge, MA: The MIT Press.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., et al. (2009). "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*. Kobe.

Sadikov, A., Možina, M., Guid, M., Krivec, J., and Bratko, I. (2007). "Automated chess tutor," in *Computers and Games, Volume 4630 of Lecture Notes in Computer Science*, Vol. 13–25, eds H.Herik, P.Ciancarini, and H.Donkers (Berlin; Heidelberg: Springer), 13–25.

Saxena, A., Driemeyer, J., and Ng, A. (2008). Robotic grasping of novel objects using vision. *Int. J. Robot. Res.* 27, 157. doi:10.1177/0278364907087172

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Netw.* 61, 85–117. doi:10.1016/j.neunet.2014.09.003

Schoner, G., and Dose, M. (1992). A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion. *Rob. Auton. Syst.* 10, 253–267. doi:10.1016/0921-8890(92)90004-I

Soetens, P. (2006). *A Software Framework for Real-Time and Distributed Robot and Machine Control*. Ph.D. thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium. Available at: http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf

Stollenga, M., Pape, L., Frank, M., Leitner, J., Förster, A., and Schmidhuber, J. (2013). "Task-relevant roadmaps: a framework for humanoid motion planning," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, Tokyo.

Stückler, J., Badami, I., Droeschel, D., Gräve, K., Holz, D., McElhone, M., et al. (2013). "Nimbro@home: winning team of the robocup@home competition 2012," in *Robot Soccer World Cup XVI* (Berlin-Heidelberg: Springer), 94–105.

Vahrenkamp, N., Wächter, M., Kröhnert, M., Welke, K., and Asfour, T. (2015). The robot software framework armarx. *Inform. Tech.* 57, 99–111. doi:10.1515/itit-2014-1066

Vahrenkamp, N., Wieland, S., Azad, P., Gonzalez, D., Asfour, T., and Dillmann, R. (2008). "Visual servoing for humanoid grasping and manipulation tasks," in *Proceedings of the International Conference on Humanoid Robots* (Daejon), 406–412.

van den Bergen, G. (2004). *Collision Detection in Interactive 3D Environments*, Taylor & Francis Group, 277.

Verschae, R., and Ruiz-del Solar, J. (2015). Object detection: current and future directions. *Front. Robot. AI* 2:29. doi:10.3389/frobt.2015.00029

Welke, K., Issac, J., Schiebener, D., Asfour, T., and Dillmann, R. (2010). "Autonomous acquisition of visual multi-view object representations for object recognition on a humanoid robot," in *IEEE International Conference on Robotics and Automation (ICRA)* (Anchorage, AK: IEEE), 2012–2019.

Zhang, F., Leitner, J., Milford, M., Upcroft, B., and Corke, P. (2015). "Towards vision-based deep reinforcement learning for robotic motion control," in *Australasian Conference on Automation and Robotics (ACRA)*, Canberra.

**Conflict of Interest Statement:** The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.