ACCESSIBLE AUGMENTED REALITY IN BLOCK-BASED PROGRAMMING ENVIRONMENTS

S. Kittani, G. Stein

Vanderbilt University (UNITED STATES)

Abstract

The integration of advanced computing concepts in introductory computer science (CS) classes is essential for fostering student interest, engagement, and retention. NetsBlox, an educational block-based visual programming environment, provides an accessible means to introduce advanced concepts to novice learners. This work introduces an extension to NetsBlox, known as ARBlox, which enhances NetsBlox with Augmented Reality (AR) capabilities.

ARBlox leverages lightweight machine-learning models to detect and track hands, faces, bodies, and AprilTags, widening the walls of potential projects while at the same time adding only a few new blocks to the environment. ARBlox empowers students to create AR projects with a webcam, enabling impactful and real-world applications on school computers. We present a set of projects that showcase ARBlox's capabilities, demonstrating its suitability for advanced CS projects while remaining novice-friendly. ARBlox empowers educators to create engaging curricula, bridging the gap between introductory and advanced CS concepts. While ARBlox aims to make CS appealing to a wider group of students by making learning more engaging and relevant, further research is necessary to empirically validate its effectiveness in achieving this goal.

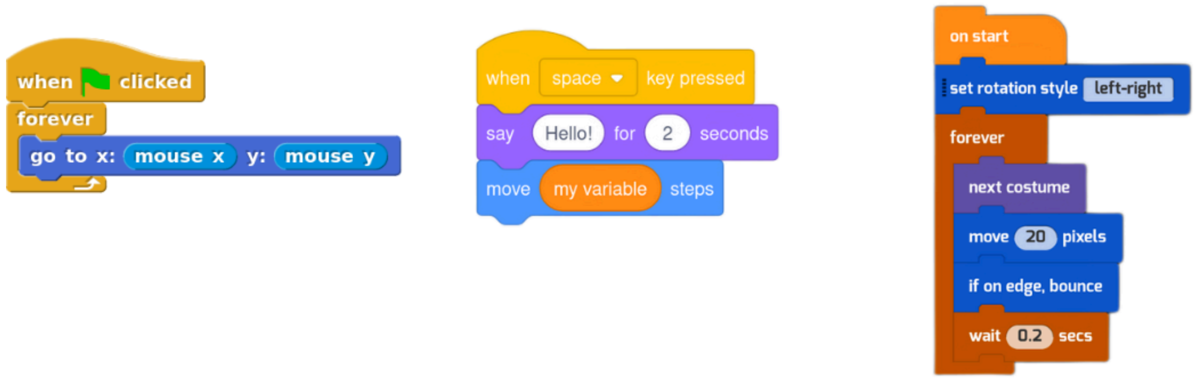
Keywords: Computer science education, augmented reality, visual programming, web browser.

1 INTRODUCTION

Augmented Reality (AR), the technology overlaying virtual objects onto the physical world, has continued to become more accessible as the hardware in common use has improved. At the same time, it has seen long-term use in education [1]. AR provides not only an interesting visual output for students, but novel methods of human-computer interaction impossible through a mouse and keyboard. The use of AR in education has included visualization, annotation, games, and storytelling [2]. Content creation through AR has generally been more accessible than the direct use of libraries to create new applications, especially for novice-level programmers. The use of AR in computer science education generally sees its use in the context of games or applications where the goal is to teach a more general set of skills, such as computational thinking or programming, with AR serving only as an exciting new interface for students to use [3]. The adoption of AR in education has spurred the development of multiple extensions to existing block-based programming environments.

Block-based programming is a “programming-primitive-as-puzzle-piece” paradigm — each command is a “puzzle-piece” that provides visual cues to guide the programmer [6]. Block-based programming environments prevent syntax errors by disallowing incompatible blocks from being joined together. Additionally, replacing text commands completely prevents syntactical errors due to spelling or punctuation. Removing these technical challenges from programming allows users to focus on developing other skills, such as computational thinking [9]. This structure makes block-based programming environments a particularly effective teaching tool for early learners [8].

Fig. 1 highlights three examples of block-based programming environments (i.e., NetsBlox, Scratch, and Tynker). These environments, as well as others, focus on providing a low floor (i.e., barrier of entry), a high ceiling (i.e., support complex projects), and wide walls (i.e., support diverse projects) [6]. However, NetsBlox, Scratch, and Tynker vary in regard to these three factors. Scratch’s target audience ranges from kids ages eight to sixteen [5]. Its simplistic design creates a low barrier of entry; however, its lack of complexity makes it unappealing to students ages 16 and up. Similar to Scratch, Tynker's target audience includes students aged five to seventeen [7]. NetsBlox has the widest target audience, with students from kindergarten to college freshman utilizing the environment [11].



```
when [flag] clicked
forever
  go to x: (mouse x) y: (mouse y)
```

```
when [space ▾] key pressed
say (Hello!) for (2) seconds
move (my variable) steps
```

```
on start
set rotation style [left-right]
forever
  next costume
  move (20) pixels
  if on edge, bounce
  wait (0.2) secs
```

Figure 1. Various block-based programming environment examples: (Left) NetsBlox, (Center) Scratch, (Right) Tynker.

Despite its success in classrooms, block-based programming poses some drawbacks. Studies have shown that students regard traditional text-based programming curricula to be significantly more relevant to real-world programming than block-based programming curricula [10]. However, students reported being much more likely to take another computer science course when learning with block-based programming rather than text-based programming [10]. To combat the former result, many block-based programming environments, including those discussed above, offer tools to widen the breadth of possible projects, engage young learners, and allow users to tinker with real-world adjacent projects.

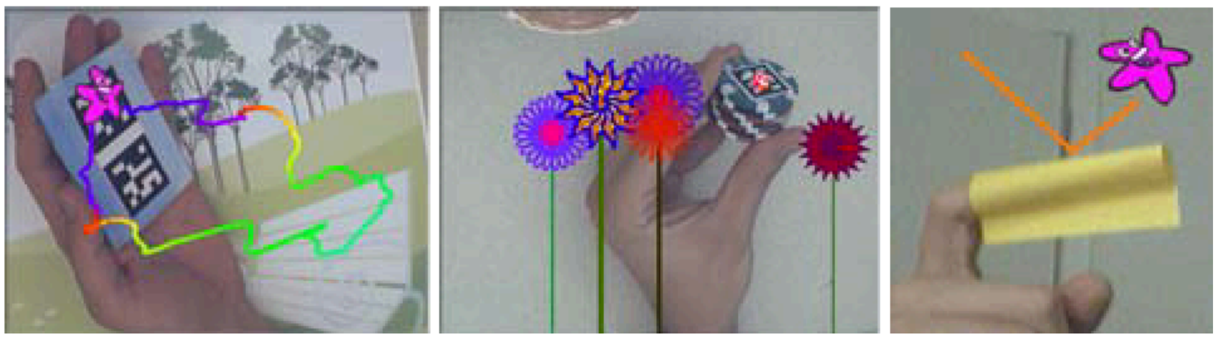


Figure 2. Projects using Augmented-Reality Scratch [4]. (Left) Drawing application, (Center) marker controlled flower game (Right) marker controlled pong game

An existing AR-based tool enabling novice students to create their own applications using AR is Augmented-Reality Scratch [4]. The tool provides support for detecting fiducial markers in the Scratch environment. The authors focused on abstracting away the complexities of the tool, aiming to appeal to a younger audience and to remain harmonious with Scratch's simplistic paradigm. Users are able to locate the center of multiple markers. Detected markers can interact with a sprite (e.g., controlling a sprite's position on the world stage). Fig. 2 includes example projects utilizing this tool.

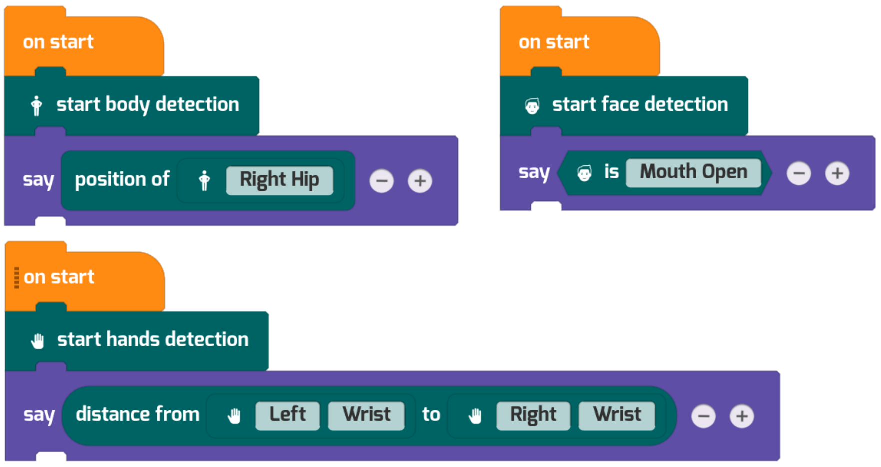


Figure 3. Tynker AR example blocks: (Top Left) Landmark coordinate retrieval. (Bottom) Landmark distance retrieval. (Top Left) Predicate retrieval.

Another pre-existing tool is Tynker's AR block library. This library consists of three main sections, providing face, hand, and body tracking (see Fig. 3). Students are able to retrieve individual landmark coordinates from the three tracking models. Additional blocks are included to help with common operations, such as calculating the distance between two landmarks. Similar to AR Scratch, the blocks abstract the handling of the model's output away from the user.

Both Augmented-Reality Scratch and the Tynker AR block library prioritize simplicity and ease of use. This focus certainly increases accessibility to young, novice users [4]. However, we argue that the focus on simplicity limits Scratch and Tynker's appeal to intermediate and advanced users. In this paper, we showcase ARBlox, an extension to NetsBlox. The ARBlox extension builds upon these tools. It provides hand, face, body, and AprilTag tracking. Additionally, it supports 3D object rendering onto AprilTags. ARBlox, in contrast to the above tools, avoids simplification of the output. By doing so, we position ARBlox to cater to older, more knowledgeable users. However, as discussed in the methodology section, this does not limit accessibility to younger populations.

2 METHODOLOGY

ARBlox is implemented as an extension to NetsBlox. ARBlox uses multiple models and algorithms to provide tracking and 3D rendering capabilities. In contrast to Tynker and Scratch, our goal with ARBlox is to appeal to learners at multiple levels. To do so, ARBlox provides significantly greater access to model output than Scratch and Tynker. For example, in Tynker's AR Block Library, the user must retrieve stage relative 2-dimensional coordinates one landmark at a time. Similarly, Scratch users must retrieve coordinates for the center of a single marker at a time. In ARBlox, however, users are provided with considerably more information. For example, when tracking a face, ARBlox returns a data structure with multiple lists of relevant data. The first list consists of every 3-dimensional landmark coordinate. The second list displays the various facial expressions alongside their respective coefficients. Throughout our implementation, we prioritize capability over simplicity. The following sections provide greater detail on the implementation of each feature.

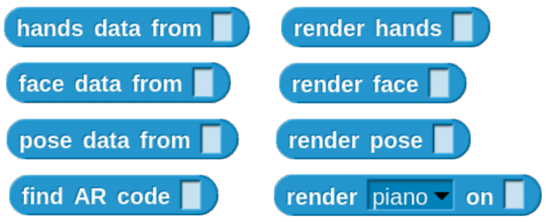


Figure 4. ARBlox blocks:

From top down: Hand Landmarker and renderer, Face Landmarker and renderer, Pose Landmarker and renderer, and AprilTag Landmarker and 3D Object Renderer.

2.1 MediaPipe

The face, hand, and body trackers use MediaPipe models for detection. MediaPipe is a suite of cross-platform perception and inference model solutions [13]. MediaPipe abstracts the complexity of both generating an effective model and integrating it with an application. Computation is done locally with light-weight and specialized models, offering low latency performance on common hardware. ARBlox specifically implements MediaPipe's Hand, Face, and Pose Landmarker models. These three models vary in complexity. However, all models have shown inference latency as low as 20 ms within ARBlox, making them ideal for projects that require real-time processing. ARBlox offers two base blocks for each MediaPipe model.

2.1.1 Hand Landmarker

The Hand Landmarker model consists of two stages. The first stage is a single-shot detection convolutional neural network (CNN) [17]. It takes an image as input. The first stage identifies and bounds a hand in the given image. The second stage, a regression based CNN, takes as input the image segment containing the hand provided by the first stage. It outputs 21 3-dimensional landmark coordinates, as well as handedness. The Z-coordinate represents relative depth in relation to the wrist. Application independent model inference latency is reported as 17ms on a Central Processing Unit (CPU) and 12ms on a Graphics Processing Unit (GPU).

ARBlox offers two blocks for hand tracking (See Fig. 4). The first, labeled "hands data from _" takes an image as input, and outputs a list. Each element represents one hand. For each detected hand, the block outputs all 21 3-dimensional landmark coordinates. Additionally, handedness is included. The user must navigate this nested data structure to retrieve information. While this implementation may not be beginner-friendly, it poses certain advantages over simplified implementations (e.g., Tynker). These advantages are discussed in the results section. In addition to the data retrieval block, ARBlox offers a simple way to visualize the hand tracker with the "render hand _" block. This block takes an image as input, overlays points and connections for landmarks, and returns the modified image. It simplifies visualization of the landmark data.

2.1.2 Face Landmarker

The Face Landmarker model pipeline includes three stages. These stages are MediaPipe BlazeFace, MediaPipe FaceMesh, and MediaPipe BlendShape. Similar to the Hand Landmarker, the first stage (BlazeFace) of this model is a CNN that detects a face in an image. It takes an image as input and then outputs the sub-regions that contain faces [19]. The second stage, a CNN, takes the BlazeFace output and identifies 478 3-dimensional landmark coordinates. The Z-coordinate is the relative depth of a landmark compared to the face's center of mass [20]. The final stage, yet another CNN, uses the 478 landmark coordinates and outputs 52 facial expression normalized prediction coefficients. Examples include cheekPuff, mouthPucker, and tongueOut [21]. The model as a whole returns the 478 landmark coordinates as well as the 52 facial expression coefficients.

Similar to the hand tracker, ARBlox offers two blocks for face tracking (See Fig. 4). The first, labeled "face data from _" takes an image as input, and outputs a list. Each element represents one detected face. Each element contains two lists. The first contains all 478 3-dimensional landmark coordinates. The second holds probability scores for 52 facial expressions. Keeping with the same paradigm, the user is responsible for retrieving data from this output. The second block, "render hand _", allows for fast visualization of the landmarks. This block is especially important for the face landmarker, given the significant effort required to visualize all 478 landmarks at once. Advantages and tradeoffs of design choices are discussed in the results section.

2.1.3 Pose Landmarker

The Pose Landmarker consists of two stages [18]. The first stage identifies a sub-region of the input image that contains a body. The sub-region that contains the body is passed to the second stage. This stage locates 33 3-dimensional landmarks [17]. The Z-coordinate represents the relative depth of a given landmark in relation to the hips. Each landmark is accompanied by a visibility probability and a presence probability. The visibility probability represents the likelihood that the landmark is visible in the input image. The presence probability represents the likelihood that the landmark is in the frame of the input image, regardless of occlusion.

Following the same pattern as the previous trackers, ARBlox offers two blocks for body tracking (See Fig. 4). The first, labeled “pose data from _”, is the main block; it takes as input an image and returns MediaPipe’s Pose Landmarker output as a nested structure. Each element in the base list represents one detected body. Each body contains all 33 3-dimensional landmark coordinates. As mentioned previously, the user is responsible for extracting data points from the data structure. The second block, “render pose _”, allows for fast visualization of the landmarks. Similar to the other render blocks, this block takes an image as input and overlays points for each landmark, as well as connections between them.

2.2 AprilTag tracking

Fiducial marker detection and pose estimation are utilized for diverse purposes, such as robotics. Fiducial markers have also been incorporated in various educational platforms, such as Scratch and NetsBlox’s RoboScape [22]. Their wide use-case applicability has led to the development of many markers, each constructed with a particular strength [29]. For example, CCTags were designed to be robust to environment lighting changes [23]. One of the most widely used fiducial markers are AprilTags. Studies have shown that AprilTags have the highest detection rate across conditions [29]. Additionally, AprilTag detection algorithms are able to process images quickly on limited hardware [24]. AprilTags, and fiducial markers in general, are often paired with pose estimation algorithms. These algorithms estimate the orientation (i.e., rotation and transform) of the marker [25]. ARBlox implements AprilTags with pose estimation for 3D object rendering. Specifically, we implement 16h5 AprilTags. The 16 represents the number of unique markers. The 5 represents the minimum Hamming distance (i.e., the difference) between any two markers. Given its simplicity, the 16h5 variation is robust to marker size and distance from the camera [14].

ARBlox implements two base blocks, an AprilTag detection block and a 3D Object rendering block. The former block, labeled “find AR code _”, takes an image as input. The block localizes AprilTags and returns the AprilTag’s ID, its Hamming distance (i.e., its error rate), and coordinates for each corner. As with the MediaPipe models, students using the base blocks must parse the output from the nested list data structure. The AprilTag detection is paired with a 3D object rendering block, labeled “render _ on _”. The first argument identifies the 3-dimensional model to render. The second argument identifies the AprilTag on which the 3D model is rendered upon. The renderer is powered by ThreeJS, a web based rendering framework [15]. ARBlox supports various built-in 3D models (e.g., cube, drum, piano, etc.).

2.3 NetsBlox Custom Block Libraries

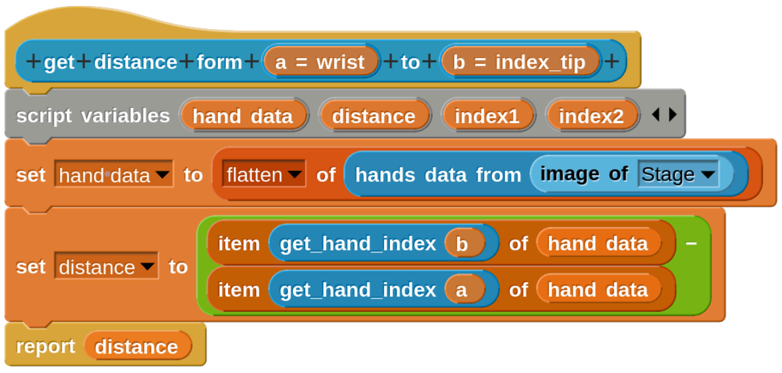

```
+get+distance+form+ a = wrist +to+ b = index_tip +
script variables hand data distance index1 index2
set hand data to flatten of hands data from image of Stage
set distance to ( item get_hand_index b of hand data ) − ( item get_hand_index a of hand data )
report distance
```

Figure 5. ARBlox custom library block that returns the distance between two hand landmarks

The provided base blocks are challenging to use, which limits accessibility to younger users. However, NetsBlox’s custom block libraries mitigate this limitation. Researchers, educators, and proficient users are able to define and share custom, scaffolded blocks with the NetsBlox community [30]. We composed such a library to make ARBlox accessible to beginners. This library contains blocks that provide coordinates for individual landmarks, detect the presence of a specific AprilTag, and compute the distance between two sets of coordinates. Fig. 5 shows an example of a custom block that returns the distance between two hand landmarks. Advanced users are encouraged to expand upon this foundation by creating new libraries. The block libraries not only provide beginners with simplified and specialized blocks, it also fosters community-based learning [26] and community-based programming [27]. Both are invaluable skills in computer science and software engineering.

3 RESULTS

ARBlox's feature set expands on Tynker and Scratch. See Table 1 for a summary table of feature comparisons. For example, Tynker offers 3 facial expression detection predicates. These include LeftEyeOpen, RightEyeOpen, and MouthOpen. By comparison, ARBlox supports 52 facial expression features. Compared to Tynker, our focus on capability of facial expression detection can make utilizing them more challenging. However, this complexity in ARBlox enables complex project possibilities. Additionally, beginner accessibility of facial expression detection is improved by custom block libraries. Furthermore, ARBlox offers an object rendering block, which allows users to overlay 3-dimensional objects into their environment. No equivalent feature is available on Tynker and/or AR Scratch.

In addition to unique capabilities, ARBlox shares many features with Tynker's AR Block Library and Augmented Reality Scratch. However, differences in implementation vary significantly for each feature, stemming from different target audiences. Our results focus on comparisons between ARBlox, Augmented-Reality Scratch, and Tynker's AR Block Library. The common difference between all comparisons is ARBlox's reduced simplification, which allows for more complex projects and extends the appeal to older, experienced users, while still providing beginner-friendly, community generated custom blocks.

Table 1. Summary comparison of available features in AR block-based programming tools.

	ARBlox	***Tynker AR [28]***	***AR Scratch [4]***
Body Tracking	✅	✅	❌
Hand Tracking	✅	✅	❌
Face Tracking	✅	✅	❌
Face Expression Detection	✅	❌	❌
Tracked Object Masking	❌	✅	❌
AprilTag Tracking	✅	❌	✅
AprilTag based 3D object Rendering	✅	❌	❌

3.1 MediaPipe Trackers

Both ARBlox and Tynker's AR Block Library provide face tracking. ARBlox uses the MediaPipe model to track the face. The model used by Tynker is unknown. However, Tynker's AR Block output indicates the use of the same MediaPipe model. ARBlox's face tracking block outputs all 478 3-dimensional landmark coordinates and 52 facial expression probabilities in a single data structure. Conversely, Tynker simplifies the model by providing 2-dimensional landmarks for a single landmark. These differences stem from the differing goals of each tool.

ARBlox's goal is to appeal to semi-experienced programmers. We aim to raise the ceiling of block-based programming projects. By providing all available data, we maximize the ratio of data per inference call. This makes ARBlox more efficient, which in turn allows for more complex projects. Meanwhile, Tynker primarily targets younger audiences. Their intuitive and simplistic face tracking implementation appeals to beginners and younger audiences. With a few blocks, you are able to create a simple program that uses a landmark. However, a program that requires 249 landmark coordinates (e.g., facial expression nearest neighbor classifier) would require 249 inference calls. In ARBlox, we would only require one. At the same time, ARblox's efficiency may hinder accessibility for beginners. This consequence is partially mitigated by NetsBlox's custom block libraries; experienced users are able to define simplified AR blocks, similar to Tynker's implementation (See Fig. 5 for an example block from our initial user-friendly library). We say partially mitigated because new users, without an instructor or guide, may have trouble finding the custom block library. In Tynker, however, the user is immediately exposed to simple blocks.

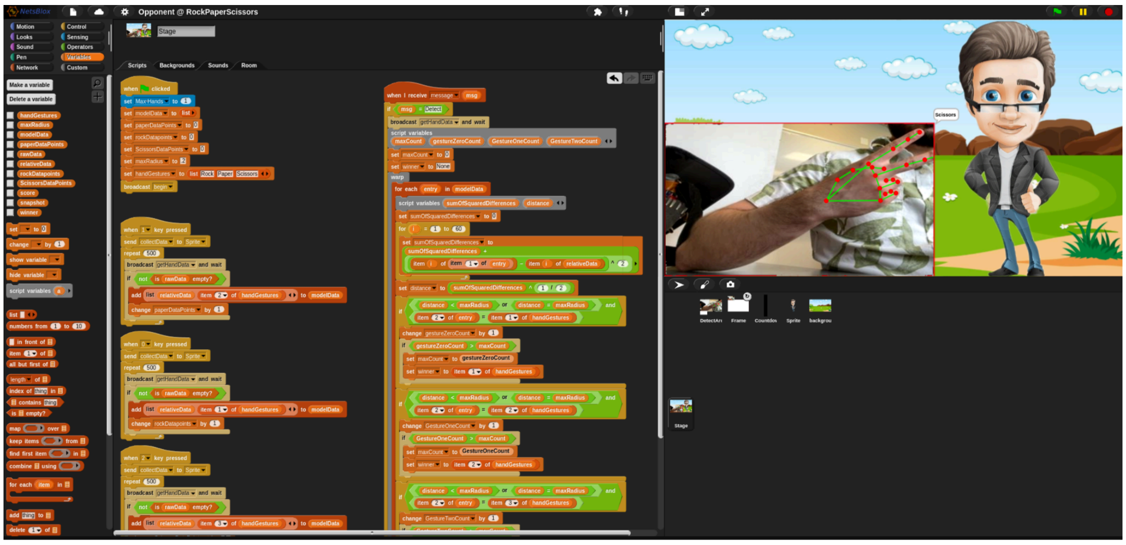


Figure 6. Nearest Neighbor implementation for hand gesture training and classification. The right shows a successful classification of the “Scissors” hand gesture.

Similarly, both ARBlox and Tynker use MediaPipe’s Hand Landmarker model to provide hand landmark coordinates for their respective environments. ARBlox’s hand tracking block maximizes data retrieval per call by returning the entire output of the MediaPipe model. Tynker, in favor of simplicity, only returns a single landmark’s coordinates per inference call. As with the face tracker, ARBlox and Tynker suffer from opposing limitations. ARBlox can be challenging to beginners. Tynker, conversely, is limited to simplistic projects.

An example of this limitation can be illustrated by considering implementations of certain advanced projects in both environments. Fig. 6 shows a Rock-Paper-Scissors ARBlox project. The project implements a nearest neighbor hand gesture classifier using ARBlox. To train the classifier, we must store hundreds of multi-dimensional data points for each hand gesture. The complexity of data collection and inference is minimized by the design of the hand tracker inference block. Each call to the model returns handedness and all landmark coordinates. These data points are flattened and stored in the appropriate gesture class. For inference, the project makes one call to the “hand data from _” block. To implement a similar project in Tynker would require an inference block for each landmark. This would result in 21 calls to MediaPipe’s Hand Landmarker model per training sample. The simplicity of Tynker makes complex projects, such as the classifier, less feasible. The same is true for complex projects incorporating the body tracking blocks.

3.2 AprilTag Tracking

The advantages of ARBlox over Augmented-Reality Scratch resemble the comparison with Tynker. In Scratch, each marker is color coded, and users are able to retrieve the center coordinates of a marker. Additionally, users can attach sprites to markers. Scratch’s implementation focuses on making fiducial marker detection as simple as possible. In contrast, ARBlox’s AprilTag implementation focuses on expanding the space of possible projects and raising the skill ceiling of NetsBlox. The AprilTag tracker returns a nested list structure that includes all detected markers in the image, as well as 4 2-dimensional corner coordinates for each marker.

Each paradigm has associated advantages and limitations. The advantage of the AR Scratch implementation is its accessibility to novice learners. Its simplistic design is intuitive and easy to work with. However, complex projects become difficult to implement by the simplicity of Scratch. This is especially true with projects that use a large number of markers. ARBlox, however, has the reciprocal advantage; its implementation offers expanded capabilities compared to Scratch, supporting complex projects and appealing to experienced programmers. This same complexity makes it less accessible for beginners. However, this limitation is mitigated by NetsBlox's custom block library, which enables user and educator defined abstractions over these complex blocks.

4 CONCLUSIONS

Advanced computing concepts, such as AR, fosters interest, engagement, and retention in novice CS courses. ARBlox, an extension of NetsBlox, is a powerful block-based AR learning tool. It provides users with powerful tracking blocks. These blocks detect hands, faces, bodies, and fiducial markers. ARBlox enables users to create advanced AR enabled applications (e.g., gesture classifiers). In contrast to Tynker and Scratch, we designed ARBlox to raise the skill ceiling by expanding the space of feasible projects. As a result, we positioned ARBlox to appeal to a more experienced, older population, relative to current block-based AR programming tools (e.g., Tynker and Scratch). We incorporate custom block libraries as a means to provide accessibility to novice learners, all while retaining a high skill ceiling. ARBlox is intended as an educational tool to teach foundational computer science concepts whilst programming with advanced, modern, and relevant technologies.

ARBlox is positioned uniquely to cater to knowledgeable users. ARBlox has the potential to engage experienced users, foster classroom engagement, and promote community learning. Despite our focus on powerful capabilities, its implementation does not raise the barrier of entry. With ARBlox, students, instructors, and independent learners are empowered to explore AR technology through an accessible, powerful environment.

4.1 Limitations and Future Directions

A major limitation of this paper is its lack of an empirical study. Testing various hypotheses would demonstrate the effectiveness of ARBlox. We propose three possible future studies that would test ARBlox's efficacy. First, comparing user preferences on AR enabled block-based programming environments. Do beginners prefer certain types of AR blocks? Do intermediate users differ in their preferences? Additionally, future studies could explore user experiences with the custom block library. Does providing custom block libraries for novice learners change their opinions on ARBlox? How does building custom block libraries change advanced user's views on the applicability of block-based programming skills? Finally, future studies could explore the effectiveness of ARBlox as an engagement tool. Do courses incorporating ARBlox have higher levels of student engagement and content retention than courses that do not utilize ARBlox? Does ARBlox appeal equally to all demographics, especially in its ability to attract underrepresented groups to computer science?

As discussed previously, a significant limitation of ARBlox is in the additional complexity it presents to the user. Whilst custom block libraries are available as a scaffold for beginners, users could have trouble finding said libraries. Future development will focus on alleviating this concern. Development efforts could link blocks with custom block libraries. Therefore, a user could easily find a custom block library that is linked to challenging blocks. Another potential feature is intelligent experience level block palettes. Future versions of NetsBlox could tailor the available blocks with regards to user reported experience levels.

In addition to the features proposed to address limitations, future research could expand ARBlox further by incorporating a means of adding custom fiducial marker libraries. There are a vast number of fiducial markers, all designed with a particular strength [29]. Adding support for custom fiducial marker libraries would further expand the space of possible projects, fostering engagement. Another future development direction will focus on expanding user control over the 3-dimensional model renderer. Future development will concentrate on enabling users to import their own model files, enabling them to not only render these models within the platform but also manipulate and interact with the rendered 3D objects extensively." These development efforts will prioritize features that raise the skill ceiling and expand the space of feasible projects.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (#1949472) and the Vanderbilt University Seeding Success Grant.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors' and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] C. Avila-Garzon, et al. "Augmented Reality in Education: An Overview of Twenty-Five Years of Research." *Contemporary Educational Technology*, vol. 13, no. 3, 2021.

[2] M.Z. Iqbal, et al. "Current challenges and future research directions in augmented reality for education." *Multimodal Technologies and Interaction*, vol. 6, no. 9, pp. 75-104, 2022.

[3] A. Theodoropoulos, and G. Lepouras. "Augmented Reality and programming education: A systematic review." *International Journal of Child-Computer Interaction*, vol. 30, 2020.

[4] I. Radu and B. MacIntyre. "Augmented-reality Scratch: a children's authoring environment for augmented-reality experiences." *Proceedings of the 8th International Conference on Interaction Design and Children*, pp. 210-213, 2009.

[5] M. Resnick, et al. "Scratch: programming for all." *Communications of the ACM*, vol. 52 no. 11, pp. 60-67, 2009.

[6] D. Weintrop. “Block-based programming in computer science education.” *Communications of the ACM* vol. 62, no. 8, pp. 22-25, 2019.

[7] E. Wafaa, and Thabet, R.A. "The Effectiveness of Tynker Platform in Helping Early Ages Students to Acquire the Coding Skills Necessary for 21st Century." *International Conference on Information Systems and Intelligent Applications*, pp. 381-397, 2022.

[8] D. Bau, e al. “Learnable programming: Blocks and beyond.” *Communications of the ACM*, vol. 60, no. 6, pp. 72–80, 2017.

[9] B. Broll, et al. “A visual programming environment for learning distributed programming.” In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pp. 81-86. 2017.

[10] D. Weintrop, and U. Wilensky. "Comparing block-based and text-based programming in high school computer science classrooms." *ACM Transactions on Computing Education (TOCE)* vol. 18, no. 1, pp. 1-25, 2017.

[11] B. Broll, et al. "Removing the walls around visual educational programming environments." In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1-9, 2021.

[12] C. Brady, et al. “Block-based abstractions and expansive services to make advanced computing concepts accessible to novices Journal of Computer Languages.” *Journal of Computer Languages*, vol. 73, 2022.

[13] C. Lugaresi, et al. "Mediapipe: A framework for perceiving and processing reality." In *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)*, vol. 2019. 2019.

[14] D. Oberkampf, et al. “Iterative pose estimation using coplanar feature points.” *Computer Vision and Image Understanding*, vol. 63, no. 3, pp. 495-511, 1996.

[15] L. Yonghuai, et al. "Teaching of advanced computer graphics with three.js." In *Proceedings of International Conference on Education and New Developments*, pp. 13-17, 2016.

[16] M. Fiala, "ARTag, a fiducial marker system using digital techniques," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 590-596 vol. 2, doi: 10.1109/CVPR.2005.74.

[17] H. Xu, et al. “GHUM & GHUML: Generative 3D human shape and articulated pose models.” In Proceedings of *the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* pp. 6184-6193, 2020.

[18] V. Bazarevsky, et al. “BlazePose: On-device real-time body pose tracking.” *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*, 2020

[19] V. Bazarevsky, et al. “Blazeface: Sub-millisecond neural face detection on mobile GPUs.” *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*. 2019.

[20] G. Yan and I. Grishchenko. “Model Card: MediaPipe FaceMesh”, 2022. Retrieved from https://storage.googleapis.com/mediapipe-assets/Model%20Card%20MediaPipe%20Face%20Mesh%20V2.pdf

[21] I. Grishchenko, et al. “Model Card: MediaPipe BlendShape V2”, 2022. Retrieved from https://storage.googleapis.com/mediapipe-assets/Model%20Card%20Blendshape%20V2.pdf

[22] G. Stein and A. Lédeczi. "Mixed reality robotics for STEM education." In *2019 IEEE Blocks and Beyond Workshop (B&B)*, pp. 49-53, 2019.

[23] L. Calvet, et al “Detection and accurate localization of circular Fiducials under highly challenging conditions. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] J. Wang and E. Olson. “AprilTag 2: Efficient and robust fiducial detection.” In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 4193-4198, 2016.

[25] Z. Zhang, et al. “Robust and Accurate Vision-Based Pose Estimation Algorithm Based on Four Coplanar Feature Points.” *Sensors*, vol. *16*, no. 12, 2173, 2016.

[26] D. A. Fields, et al. “Youth computational participation in the wild: Understanding experience and equity in participating and programming in the online scratch community.” *ACM Transactions on Computing Education (TOCE)*. vol. 17, no. 3, pp. 1-22, 2017.

[27] A. Anderson, et al. “Discovering value from community activity on focused question answering sites: a case study of stack overflow.” In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 850-858, 2012.

[28] Tynker. “Coding for kids and teens made easy.” Retrieved from https://www.tynker.com/

[29] M. Kalaitzakis, et al. “Fiducial markers for pose estimation: Overview, applications and experimental comparison of the ARTag, AprilTag, ArUco and STag markers.” *Journal of Intelligent & Robotic System.* vol. 101, pp. 1-26, 2021.

[30] G. Stein. *A Novice-Friendly Networked Educational Robotics Simulation* [PhD Dissertation], Vanderbilt University*,* 2024.